Asthma Monitoring

Aventí Bonson

June 2011

Acknowledgement

This thesis would not have been possible without the invaluable help and advises of Prof. Tomasz Zieliński, who guided and encouraged me throughout all my research. I am also very thankful to my colleagues and friends Marcin Wiśniewski, Krzysztof Łoziak, Marek Sikora, Jacek Dańda, Pawel Cul and Marcin Barylski involved in the eHeath global system. It was a pleasure to work with them.

I would also like to show my gratitude to my family for their support and help during my stay in Poland.

Finally, I have to thank all my friends from Kraków, Justyna and my friends from Spain who have visited me. They made this experience unforgettable.

Contents

1 Introduction **10**

1.1 Mobile Development in Medical Issues 11

1.2 Goal of the Work 12

1.3 Overview of the Report 13

2 eHealth **14**

2.1 The Beginning of eHealth 14

2.2 eHealth Networks 16

2.3 Medical Aspects Related with Asthma and COPD Diseases 17

2.4 Asthma and COPD Monitoring 20

3 Windows Mobile Development **23**

3.1 Programming Language and Environment 23

3.1.1 Tools Required 24

3.1.2 .Net Compact Framework Platform 25

3.1.3 C# Programming 26

3.1.4 Terminals Working under Windows Mobile 6.5 Software 27
3.2 Getting Started with Windows Mobile Application Development 28
3.2.1 Creating, Building and Running an Application 29
3.2.2 Graphical Interface and Programming Side 29

4 Asthma Monitoring Global Network **31**
4.1 System Architecture 32
4.1.1 Overall Architecture 32
4.1.2 Networking Aspects 33
4.2 Communication Scenarios 34
4.2.1 Databases. eHealth Server 35
4.2.2 Lung Efficiency Measurements 36
4.2.3 Wheezes Detection 37
4.2.4 Air Quality Monitoring 38
4.2.5 Consultation 39
4.3 Data Specification 40
4.3.1 Input Data 40
4.3.2 Output Data 41

5 AsthmaApp **42**
5.1 AsthmaApp Architecture 43
5.2 Medical Requirements 44
5.2.1 Daily Test Questions 44

5.2.1.1 Morning Questions 45
5.2.1.2 Evening Questions 45
5.2.1.3 On Demand Questions 46
5.2.1.4 Questions Punctuation and Correlation with Number of Inhalations 46
5.2.2 PEF/FEV1 Values 48
5.2.3 Setting a Final Health State 50
5.3 Graphic Design and Forms 50
5.3.1 Graphic Design 51
5.3.2 Forms Management 52
5.3.3 AsthmaApp Components 52
5.3.4 First Run and Welcome Form 53
5.3.5 Application Modes 53
5.4 AsthmaApp Data 54
5.4.1 Data Structure 55
5.4.1.1 Results Sent to the Server 55
5.4.1.2 .wav Files Format 56
5.4.2 Application Files 56
5.4.2.1 Morning/Evening Files 57
5.4.2.2 PEV/FEV1 Files 58
5.5 Connectivity 58
5.5.1 Internet Connection 58

5.5.2 Bluetooth Connection 59
5.5.3 GPS 61
5.5.4 SMS Alerts 63
5.6 Daily Test Forms 63
5.6.1 Daily Tests Algorithms 63
5.6.2 Taking a Daily Test 65
5.7 Pollution Alerts Form 66
5.8 Data Visualization Forms 67
5.8.1 Using PocketGraphBar.dll 68
5.8.2 Charts Data Processing 69
5.9 Edit Patient Information Form 70
5.10 Settings Form 72
5.11 Additional AsthmaApp Aspects 73
5.11.1 AsthmaApp Icon 73
5.11.2 Automatic Touchable Keyboard 74
5.11.3 Programming Graphics 75

6 Conclusions **76**

6.1 Summary and Conclusions 76
6.2 Future Research 77

List of Figures

2.1 Generic eHealth network. MD-medical doctor, DB-data base 16

2.2 Asthma remote monitoring service. MD-Medical doctor. 21

3.1 Windows Mobile logo 24

3.2 System architecture including .NET Compact Framework 26

3.3 HTC HD Mini 28

3.4 Part of design view in Visual Studio for Windows mobile development . 30

4.1 Asthma network data flow 33

4.2 Wheezes recorder 37

4.3 Air Pollution Network 39

5.1 AsthmaApp architecture 43

5.2 Main form visual design 51

5.3 AsthmaApp modes (default on right, children on center and elderly people on left) 54

5.4 Accessing files algorithm 64

5.5 Pending test results 66

5.6 Last week visualization. 70

5.7 On the left first run form. On the right edit patient information form. . 71

5.8 Settings form 72

5.9 AsthmaApp icon 74

List of Tables

2.1 Standard PEF values for males 18

2.2 Standard PEF values for females 19

2.3 Standard FEV1 values for males 19

2.4 Standard FEV1 values for females 20

4.1 Input data stored in Medical Data Bases 40

4.2 Output data in Medical Data Bases 41

Abstract

The asthma disease is a specific lung chronic illness which is hard, or even impossible to cure. It consists in the inflammation of the airways which causes spasms and swells periodically. The sufferer then must wheeze or gasp for air. Its therapy is highly related to a proper patient monitoring and restricted to inform the individual in case of symptom intensification as well as recommending appropriate medicines. In this report, we firstly introduce the reader to an innovate architecture for a patient's monitoring system in the asthma diseases field. The system is mainly composed by medical devices such as sensors and recorders, mobile terminals and external servers in order to obtain a platform able to perform a remote monitoring treatment of the patient.

However, the aim of this study is to develop a smartphone application, which is based on a periodical tests to the patient composed by short questions and data collected by the medical sensors. After being processed the information and, depending of the results obtained, the application is recommending to the patient what to do, as well as giving a visual information in a clear way about the patient's current health state. Additionally, there is an option to record the patient's breathing in each test, which allows the doctors to proceed an accurate remote analysis in order to detect anomalies. Furthermore, the application pretends to be a smart tool able to detect possible health ricks such as air pollution or risk of asthma attack. The application also displays data reordered in the daily tests, allowing the patient to check the progress of the illness from previous days up to the last year.

AsthmaApp is an application developed within the eHealth framework, with the purpose of make the life of asthma sufferers easier, avoiding periodical visits to the hospital. The results obtained demonstrate that it is viable to use and commercialize that technology in the near future.

This thesis has been developed under the support of european grand called Future Internet Engineering[WebPjct]. The project covers the development and testing of infrastructure and services for the future generation Internet, where eHealth Networks are a revenant part.

Chapter 1

Introduction

Remote monitoring eHealth systems are becoming powerful tools in the medical field. The main principle of the eHealth systems is to take vital signals from or around the patient and to send them to an external medical database in order to be processed. Medical specialists can make up a prescription based on actual results from sensors and on a previous medical history saved in digital databases. Then, it is possible to send the diagnosis to the patient and start the pertinent treatment. One of the most important requirements for delivery of such systems it is the ubiquity. eHealth services have to be available as much as possible, anywhere and anytime. Nowadays, incoming eHealth systems are able to control cardiac, diabetic and pulmonary diseases successfully. Unfortunately, systems built for cardiac and diabetic diseases are more common than pulmonary monitoring.

The World Health Organization [ASTH] published that around 300 million people are suffering asthma disease[Web0]. In 2005 died around 255 000 people. It is estimated that by 2025, there will be around 100 million asthma sufferers more than nowadays. It is also important to take into account that for every 250 deaths in the World, one of them is directly caused by asthma. These pieces of information show the magnitude of the problem, and how much relevant will be in the near future pulmonary remote monitoring eHealth systems. The all time monitoring leads to increment the patient's comfort and life quality as well as it makes the illness less troublesome. This report will do an approach to a eHealth monitoring system, developing and testing a Smartphone application, which will be able to interact with the whole eHealth network. Finally, the system will be fully operative and able to be used by patients and doctors, as it will be explained in the following chapters.

1.1 Mobile Development in Medical Issues

Nowadays, by grace of new technology improvements, we are able to achieve things that would be unthinkable few years ago. This steady progress has allowed to expand fields such as mobile technology, which is currently able to interact with other fields, beforehand distant, such as medicine. In this way, several companies and researchers are developing Smartphone applications in order to cover a large range of diseases, with the aim to improve life quality of the sufferers.

However, not all medical applications which can be found in the market or under development are based on the treatment of diseases. There are also a large number of applications focused on the diseases prevention and healthy lifestyle. In this context, those applications are mostly commercial aimed and there is no direct interaction between patients and doctors. Nonetheless, some aspects of health and fitness are significantly improved by using medical applications. In this way, it is asserted that they are an important tool to prevent future illness. The most relevant and innovative fields that may be covered by those applications are explained in the list below:

- **Sleep monitoring**: helping the user by snoring and movements on the bed recordings in order to monitor the sleep and determine in which sleep phase is the patient. Then, the application wakes the user in the lightest sleep phase.
- **Losing weight**: monitoring and tracking the user daily meals and activities, providing databases with different kind of healthy meals. In addition, the users are able to check the results and improve them weight as well as some applications are giving advices in order to change wrong habits.
- **Period monitoring**: allowing to predict next menstrual periods by taking data from the user. When data is taken, the applications alert the user by a set of alarms. It is also used to provide charts, showing last periods progress and other useful data.
- **Pregnancy monitoring**: keeping track of the user pregnancy states and also providing weekly tips. Those applications are displaying calendars with all the pregnancy phases, and even telling important data such as the approximate size of the baby, the progress of the pregnancy and alerts in case of danger. Finally, some of them have an option of playing a list of songs to relax the user and the baby.

- **Heart beating**: they are mainly used to optimize physical exercises and to track progressions as well as alerting the users in the case of danger. In some cases, those applications might be used to monitor the beating of the users with heart diseases.
- **Brain training**: designed for elderly people with memory problems. Those applications help to stimulate the mind of the users by a periodical games or tests of concentration, memory, logic etc. The difficulty fits to the users improvements with the aim of increase the training and achieve every day better results.
- **Health care**: it is easy to find a lot of applications related to health care. Among them, it is important to emphasize applications based on relaxing exercises, fitness, sports and healthy activities. The main purpose is to improve the users fitness form, and consequently them wellness.

On the other hand, there is also a huge development of applications in the eHealth's frame, which it is targeted at customizing the treatment for specific diseases and allowing the direct interaction between patients and doctors. eHealth applications are developed under doctors policies, and they should only be used by patients. Unfortunately, eHealth applications are not in an advanced stage of development, and currently it is very difficult to find hospitals equipped with such technology. Taking into consideration the fast developing of eHealth technologies, it is sure that shortly it will be commonplace. The field of eHealth is being widely studied, taking importance in medicine. Furthermore, new applications are appearing as a result of the long research in this field. At present, firsts patients are benefiting from this new technology.

1.2 Goal of the Work

The main purpose of the carried out work is to develop an asthma medical application for Smartphone using Visual Studio and Windows Mobile Developing Tools in order to be integrated in a whole eHealth network. AsthmaApp pretends to be an application able to interact between patients and the Hospital Server sending data in two ways and allowing a full disease monitoring. The application is aimed at the

Asthma and **COPD**[1], avoiding periodical visits to the hospital. AsthmaApp is based on simple daily tests and wheeze recordings in order to set the state of the patient and then, to inform the patient and doctors. Therefore, to lead to our objective, different modules are programed such as **Bluetooth Connection**, **Server Internet Access** (switching between Wifi or 3G), **GPS Connection**, **Sending Sms Code** and **Graphical Data Visualization**. Thus, aggregating all modules it is obtained the main program. In the chapter number 5, the structure of the AsthmaApp will be accurately explained, showing the roles of the different modules and the relevance of each one. Moreover, a clear graphical interface has been designed. It is composed by three different application modes, with the aim of making accessible the application to everybody. Consequently, it is possible to select between Default, Children and Elderly People modes depending of the user's needs. The results obtained demonstrate that the application interacts successfully with the eHealth network. The ideal final work, which will be done in the next months, would be to test the whole eHealth system with real patients, starting a remote monitoring and treating of the disease.

1.3 Overview of the Report

This report starts with an introduction to eHealth giving information about its meaning, history, frameworks and monitoring as well as a necessary explanation of the medical aspects and values related with the asthma disease. The third chapter is focused on the Windows Mobile development. In this chapter the reader will be introduced into the programming language and environment necessary to develop an application under Windows Mobile 6.5. Furthermore, an approach to a HTC Mini HD (device used in this report) will be done. Chapter 4 deals with the whole eHealth Network showing its structure, data specifications and application scenarios. After all, the reader will come across the main chapter of the report. This chapter discuses the AsthmaApp architecture and features, getting straight to the point of programming issues necessaries to understand all the AsthmaApp's functions. Finally, the last chapter makes a short summary of the work and discusses some aspects related with further researches.

[1] COPD: Chronic Obstructive Pulmonary Disease.

Chapter 2

eHealth

eHealth, or also written e-Health, is a relatively new term needed to describe the use of new communication technologies in the medicine and healthcare fields. Nowadays, healthcare practices have undergoing relevant changes by using modern electronic processes and ways of communication, making the actions of to get (or to record), to transmit and to store the data electronically, and also sometimes wirelessly. This step towards a new conception of healthcare is implying a deep change in the health sector, taking into consideration the recently important role of Internet and remote health monitoring in medicine.

2.1 The Beginning of eHealth

The term eHealth was created at the end of nineties. At that time, the results of studies about health and technology showed that rate cost-effectiveness of telemedicine[1] and telehealth[2] improves considerably when they are part of an integrated use of telecommunications and information technology. This principle implied the death of telemedicine as a specific field because telemedicine by itself solely refers to the use of communications and information technologies for the delivery of clinical

[1] Telemedicine: physical and psychological treatments at a distance.
[2] Telehealth: telemedicine using telecommunication technologies.

care. The same could also be said to many other fields in medical informatics technologies such as electronic patient records and information systems. eHealth represents itself as a common name which includes all technological fields. Finally, few years later, the term eHealth was consolidated and suggests that the integrated-healthcare-systems' properties, possibilities, and consequences are more than the sum of the single-component outcomes.

Thus, it is said that the eHealth term contains a huge range of medical and technological fields, as it is shown in the following list:

- **Electronic health records.**
- **Telemedicine.**
- **Consumer health informatics**: usage of electronic resources and technologies on medical issues.
- **Health knowledge management:** a range of strategies used in an organization in order to identify, create, represent, distribute, and enable adoption of insights and experiences in the health sector.
- **Virtual healthcare teams:** consisting of healthcare professionals who collaborate and share information on patients through digital equipment.
- **mHealth (mobile health):** term used for the practice of medical and public health, supported by mobile devices.
- **Medical research powered by Grids:** combination of computer resources to handle large amounts of data.
- **Healthcare Information Systems:** medical software dealing with resources, devices, and methods required to optimize the acquisition, storage, retrieval, and use of information in the health sector.

2.2 eHealth Networks

eHealth networks have as main benefits the improvement of the care quality and the reduction of healthcare costs. Those benefits are also coupled with high levels of patient satisfaction and wellness. The basic service offered by eHealth networks consists of three layers: sensing, analysis and service. The sensing layer is the responsible for measuring health condition of people, recognizing living patterns such as food and exercise, and measuring patient surroundings. The analysis layer receives data from the first layer in order to analyze them. The service layer provides qualified people with intelligent eHealth services using the data previously stored in the analyzing layer. Networks layers are accurately explained in the literature [JJP09].

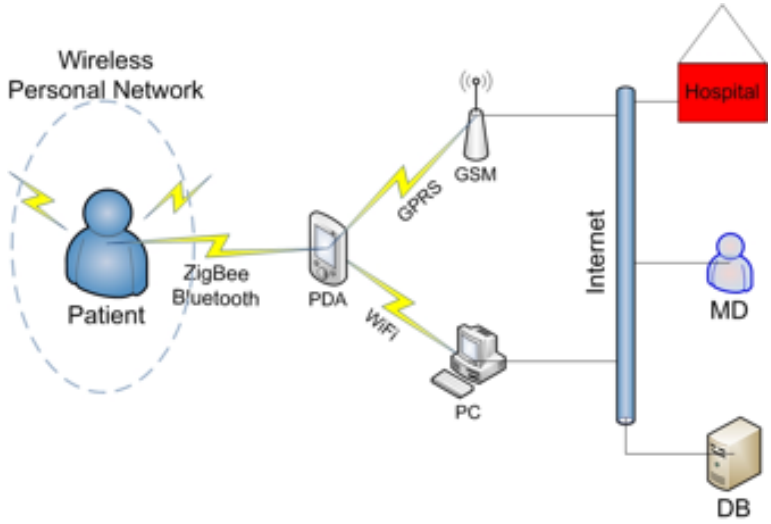

Figure 2.1: Generic eHealth network. MD-medical doctor, DB-data base

In the left side of the figure 2.1 the sensing layer is shown. The patient is surrounded by Wireless Personal Network, which contains generally the medical sensors required to take the patient vital signs[DDS09]. Measurements taken by the sensor are firstly sent to the personal server (Smartphone or PDA) via ZigBee[3] or, in the majority of the cases, via Bluetooth. Data may be analyzed locally by the Smartphone or remotely being sent directly to the hospital server. Once processed the data, it is sent to the third layer (hospital side in the right of the figure) via WIFI/3G or GPRS. The usual

[3]ZigBee: specification for a suite of high level communication protocols using small, low-power digital radios.

configuration corresponds to analyze the main part of the data in the hospital side, where a diagnosis can be done by qualified personnel such as doctors, nurses and other specialists. At last, data are stored in the data bases. The full two-way communication allows the possibility of an answer from the hospital side in the case of alert or just as a response treatment.

eHealth networks can be divided in two different groups: wireless and wired networks. Wireless networks have some important benefits compared to wired networks. First of all, wired cables connected between devices significantly restrict the patient mobility as well as reduce the patient's contort. They also may obstruct the correct usage adding an extra complexity to the system. Wired systems do not allow the users to monitor themselves during daily living. On the other hand, wireless networks allow the patients to lead a comfortable living without consciousness and limitations on wired cable. Nowadays, wireless networks are commonly accepted and used in almost all scenarios.

2.3 Medical Aspects Related with Asthma and COPD Diseases

In the context of asthma remote monitoring, there are few medical values necessaries to predict possible asthma attacks. Although not all breathing difficulties necessary indicate a disease, those values provide and evidence about the health-state of the patient. The airflow obstruction caused by the disease of the airways can be measured by a group of medical values, including:

- **Forced Expiratory Volume (FEV):** maximum volume of air that can be forced out taking a deep breath.
- **FEV1**: the forced expiratory volume of air in the first second.
- **Peak Expiratory Flow (PEF):** the maximum rate of airflow achieved during expiration.

Taking into consideration the opinion of the doctors[4], values used to analyze the patient care in AsthmaApp system are Peak Expiratory Flow and Forced Expiratory Volume 1. The PEF and FEV1 results are obtained without participation of doctors, using a medical device called spirometer[5] directly by the user. "*The method designed to make spirometry test is based on the calculation of the total respiratory input impedance as the ratio between the pressure measured at the patient's mouth and the volumetric flow rate while the system is stimulated with a pressure stimulus at specific frequencies*"[GMG09]. By using a spirometer, it is easy to allow a remote monitoring of asthma and COPD. The only drawback is the current high price of those devices in the market.

Furthermore, it exists a predicted normal values of PEF and FEV1 depending on the age, sex and height of the patient. They are also called standard PEF and FEV1 values. In the following tables the values used by AsthmaApp are shown for males and females:

MALE		Height																		
		100	105	110	115	120	125	130	135	140	145	150	155	160	165	170	175	180	185	190
Age	**5**	24	51	77	104	130	156	183	209	236	262	289	315	342	368	368	421			
	8	24	51	77	104	130	156	183	209	236	262	289	315	342	368	368	421			
	11	24	51	77	104	130	156	183	209	236	262	289	315	342	368	368	421			
	15									414	423	432	440	448	456	456	469	476	482	488
	20									456	466	475	484	492	500	500	515	522	529	536
	25									481	491	501	510	519	527	527	543	551	558	564
	30									494	504	514	524	533	542	542	558	566	573	580
	35									499	509	519	529	538	547	547	563	571	578	585
	40									497	508	518	527	536	545	545	561	569	576	583
	45									491	501	511	520	530	538	538	554	562	569	576
	50									480	491	500	510	519	527	527	543	550	557	564
	55									467	477	487	496	505	513	513	528	836	543	549
	60									452	462	471	480	489	497	497	512	519	525	532
	65									436	445	454	463	471	497	497	493	500	506	513
	70									418	427	436	444	452	460	460	474	480	486	492
	75									400	408	417	425	432	440	440	453	459	465	471
	80									381	389	397	405	412	419	419	432	438	444	450
	85									362	370	378	385	392	399	399	411	417	422	428

Table 2.1: Standard PEF values for males

[4]During the development of the system, it has been a close cooperation with pulmonary diseases specialists from the Hospital Jana Pawla II, Kraków.

[5]Spirometer: device for measuring the volume of air inspired and expired by the lungs.

FEMALE		Height																			
		100	105	110	115	120	125	130	135	140	145	150	155	160	165	170	175	180	185	190
Age	**5**	39	65	92	118	145	171	197	224	250	276	303	329	356	382	408	435			
	8	39	65	92	118	145	171	197	224	250	276	303	329	356	382	408	435			
	11	39	65	92	118	145	171	197	224	250	276	303	329	356	382	408	435			
	15									348	355	360	366	371	376	381	385	390	394	398
	20									369	376	382	388	393	398	403	408	413	417	421
	25									380	387	393	399	405	410	415	420	425	429	433
	30									384	391	397	403	409	414	419	424	429	433	438
	35									383	390	396	402	408	413	418	423	428	432	436
	40									379	385	391	397	403	408	413	418	423	427	432
	45									371	378	384	390	396	401	406	411	415	419	424
	50									362	369	375	381	386	391	396	401	405	409	414
	55									352	358	365	370	375	380	385	389	394	398	402
	60									340	347	352	358	363	368	372	377	381	385	389
	65									328	334	340	345	350	355	359	364	368	372	375
	70									316	321	327	332	337	341	346	350	354	358	361
	75									302	308	313	318	323	327	331	335	339	343	347
	80									289	294	300	304	309	313	317	321	325	328	332
	85									276	281	286	290	295	299	303	307	310	314	317

Table 2.2: Standard PEF values for females

MALE		Height																		
		100	105	110	115	120	125	130	135	140	145	150	155	160	165	170	175	180	185	190
Age	**5**	2.275	2.447	2.575	2.791	2.963	3.135	3.307	3.479	3.651	3.823	3.995	4.167	4.339	4.511	4.683	4.855			
	8	2.176	2.348	2.520	2.692	2.864	3.036	3.208	3.380	3.552	3.724	3.896	4.068	4.240	4.412	4.584	4.686			
	11	2.077	2.249	2.421	2.593	2.765	2.937	3.109	3.281	3.453	3.625	3.797	3.969	4.141	4.313	4.485	4.657			
	15									3.321	3.493	3.665	3.837	4.009	4.181	4.353	4.525	4.697	4.869	5.041
	20									3.156	3.328	3.500	3.672	3.844	4.016	4.188	4.360	4.532	4.704	4.876
	25									2.991	3.163	3.335	3.507	3.679	3.851	4.023	4.195	4.367	4.539	4.711
	30									2.826	2.998	3.170	3.342	3.514	3.686	3.858	4.030	4.202	4.374	4.546
	35									2.661	2.833	3.005	3.177	3.349	3.521	3.698	3.865	4.037	4.209	4.381
	40									2.496	2.668	2.840	3.012	3.184	3.356	3.528	3.700	3.872	4.044	4.216
	45									2.331	2.503	2.675	2.847	3.019	3.191	3.363	3.535	3.707	3.879	4.051
	50									2.166	2.338	2.510	2.682	2.854	3.026	3.198	3.370	3.542	3.714	3.886
	55									2.001	2.173	2.345	2.517	2.689	2.861	3.033	3.205	3.377	3.549	3.721
	60									1.836	2.008	2.180	2.352	2.524	2.696	2.868	3.040	3.212	3.384	3.556
	65									1.671	1.843	2.015	2.187	2.359	2.531	2.703	2.875	3.047	3.219	3.391
	70									1.506	1.678	1.850	2.022	2.194	2.366	2.538	2.710	2.882	3.054	3.226
	75									1.341	1.513	1.685	1.857	2.029	2.201	2.373	2.545	2.645	2.889	3.061
	80									1.176	1.348	1.520	1.692	1.864	2.036	2.208	2.380	2.552	2.724	2.896
	85									1.011	1.183	1.355	1.527	1.699	1.871	2.043	2.215	2.387	2.559	2.731

Table 2.3: Standard FEV1 values for males

FEMALE		Height																			
		100	105	110	115	120	125	130	135	140	145	150	155	160	165	170	175	180	185	190
Age	**5**	1,53	1,66	1,8	1,93	2,06	2,2	2,33	2,46	2,6	2,73	2,87	3	3,13	3,27	3,4	3,53			
	8	1,45	1,58	1,71	1,85	1,98	2,11	2,25	2,38	2,51	2,65	2,78	2,91	3,05	3,18	3,32	3,45			
	11	1,36	1,5	1,63	1,76	1,9	2,03	2,16	2,3	2,43	2,56	2,7	2,83	2,96	3,1	3,23	3,36			
	15									2,32	2,45	2,59	2,72	2,85	2,99	3,12	3,25	3,39	3,52	3,65
	20									2,18	2,31	2,45	2,58	2,71	2,85	2,98	3,11	3,25	3,38	3,51
	25									2,04	2,17	2,31	2,44	2,57	2,71	2,84	2,97	3,11	3,24	3,37
	30									1,9	2,03	2,17	2,3	2,43	2,57	2,7	2,83	2,97	3,1	3,23
	35									1,76	1,89	2,03	2,16	2,29	2,43	2,56	2,69	2,83	2,96	3,09
	40									1,62	1,75	1,89	2,02	2,15	2,29	2,42	2,55	2,69	2,82	2,95
	45									1,48	1,61	1,75	1,88	2,01	2,15	2,28	2,41	2,55	2,68	2,81
	50									1,34	1,47	1,61	1,74	1,87	2,01	2,14	2,27	2,41	2,54	2,67
	55									1,2	1,33	1,47	1,6	1,73	1,87	2	2,13	2,27	2,4	2,53
	60									1,06	1,19	1,33	1,46	1,59	1,73	1,86	1,99	2,13	2,26	2,39
	65									0,92	1,05	1,19	1,32	1,45	1,59	1,72	1,85	1,99	2,12	2,25
	70									0,78	0,91	1,05	1,18	1,31	1,45	1,58	1,71	1,85	1,98	2,11
	75									0,64	0,77	0,91	1,04	1,17	1,31	1,44	1,57	1,71	1,84	1,97
	80									0,5	0,63	0,77	0,9	1,03	1,17	1,3	1,43	1,57	1,7	1,83
	85									0,36	0,49	0,63	0,76	0,89	1,03	1,16	1,29	1,43	1,56	1,69

Table 2.4: Standard FEV1 values for females

Standard values are compared with the real values taken by the spirometer and then, following the directives of the doctors, a health state is set. After each spirometry, the patient receives information about his health status, therapy directions and cure doses. The whole algorithm setting the final patient state will be accurately explained in the chapter number 5.

2.4 Asthma and COPD Monitoring

Home Asthma and COPD monitoring provides assistance to the patients in following self-care treatment prescribed by their doctors, and notifies to health care personnel whether any negative trends are detected. This system allows an early detection of potentially dangerous situations in order to proceed a medical intervention as soon as possible. In recent years, patient self-management is mainly composed of the following actions:

- **Medical test:** consisting in series of questions related to breathing and coughing. The questions are usually given by the terminal (Smartphone) and they differ depending on the moment of the day (usually morning and evening). Finally, data are sent to the medical DataBase via Wifi/3G.

- **Expiratory flow and volume monitoring (PEF and FEV1):** as it was introduced in the section above, the patient uses a spirometer in order to take a pulmonary test. Data obtained are sent to the terminal via bluetooth, or just input as a number by the patient.
- **Chest auscultation (wheezes[6] detection):** the patient records the lung sounds using a recorder. Recorders are usually composed of microphones attached to stethoscopes, which are fixed in different parts of the patient's chest. The aim of this test is to analyze the data in hospital side, trying to find wheezes evidences, which are a clear symptom of risk alert. It is used to send the recorded wave files to the Smartphone via bluetooth and then, to the hospital DataBase.

Remote monitoring services may include other functions such as air pollution monitoring. Ambient monitoring provides information about the air pollution concentration, necessary data in some scenarios because it may be a direct cause of patient's health deterioration. External servers provide checking air quality services and the patient is able to have on demand access to the current air pollution state.

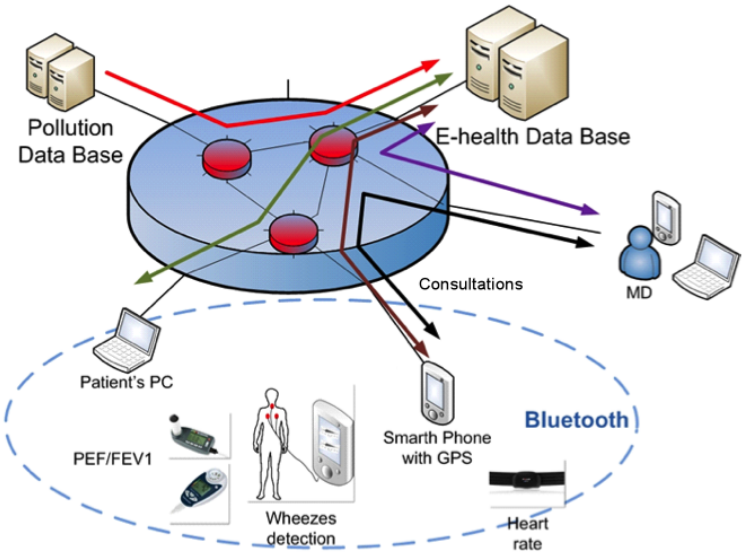

Figure 2.2: Asthma remote monitoring service. MD-Medical doctor.

[6]**Wheeze: is a continuous, coarse, whistling sound produced in obstructed respiratory airways during breathing.**

Figure 2.2 shows the main components of Asthma remote monitoring system as well as typical data flow between different parts of the system. Present systems use bluetooth as a communication channel in the Personal Area Network. The application's data flow starts from the user side after doing a request. eHealth networks should ensure safe and lossless connection between main modules. The application of asthma monitoring usually allows bidirectional connection between patient, e-health server and medical doctors. At last, communications between eHealth Servers and pollution data bases are periodical, uploading air pollution information every time interval previously programed by the Hospital Sever side.

Chapter 3

Windows Mobile Development

In order to develop an application, it is needed to decide which mobile platform will be used. A mobile platform is an operating system responsible for control terminal's hardware. Nowadays, there are many mobile platforms in the market such as Windows Mobile, Windows Phone, Android, iOS, Symbian and BlackBerry. The purpose of this report is to develop an application under **Windows Mobile**, which offers touchscreen big support, touch control, consistent navigation and responsive user interface among other interesting features. The Smartphone given and used in this report is the **HTC HD mini**. Nevertheless, AsthamApp works under all Windows Mobile terminals.

3.1 Programming Language and Environment

Windows Mobile 6 is a mobile operating system developed by Microsoft. It is made for Smartphones, PDAs and mobile devices. The current last version, which will be used in this report, is Windows Mobile 6.5. Version 6.5 is an upgrade to Windows Mobile 6.1 that was released to manufacturers in May 2009, and the first devices running the operating system debuted in late 2009. Next sections will explain tools required as well as programming language in order to contextualize steps and previous knowledge needed to create, to develop and to run an application under Windows Mobile.

Figure 3.1: Windows Mobile logo

3.1.1 Tools Required

To get started with Windows Mobile Applications Development, it is necessary to install a set of programming and management tools:

- **Microsoft Visual Studio 2008 Professional Edition**: is a comprehensive set of tools that accelerates the process of turning the developer's vision into reality. Visual Studio 2008 Professional Edition is engineered to support development projects that target the Web and devices (including ASP.NET[1] AJAX), Windows Vista, Windows Server 2008, the 2007 Microsoft Office system, SQL Server 2008, and Windows Mobile devices (including .NET Compact Framework[2]). It is the main tool necessary to start developing mobile applications.
- **Windows Mobile 6 Professional Software Development Kit (SDK):** adds documentation, sample code, headers, library files and emulator tools to Visual Studio 2008 that let the developer build applications for Windows Mobile 6. Professional version is necessary to develop applications to Smartphones with touch screen.
- **Windows Mobile 6.5 Professional Developer Tool Kit:** adds documentation, sample codes and libraries to Visual Studio as well as new emulators and features such as gestures. It is necessary to developers who need to program for WM 6.5 OS and to use its new features. It can be considerated an expansion development software, so Windows Mobile 6 SDK must be firstly installed.
- **Windows Mobile Device Center 6.1:** is a tool required to synchronize Windows Mobile devices with personal computers. This tool is necessary to run applications in professional devices, avoiding the usage of emulators.

[1]ASP.NET: is a web application framework developed and marketed by Microsoft, allowing programmers to build dynamic web sites, web applications and web services.

[2].NET Compact Framework: contains the common language runtime and class libraries built for develop Mobile Applications

3.1.2 .Net Compact Framework Platform

The .NET Framework is a Microsoft's platform designed to build applications that have visually stunning user experiences, improved and secure communication, and the ability to model variate business processes. The .NET Framework has as a relevant features:

- **Common Language Runtime:** provides an abstraction layer over the operating system.
- **Base Class Libraries:** pre-built code for common low-level programming tasks.
- **Development frameworks and technologies:** reusable and customizable solutions for larger programming tasks.

Furthermore, .NET Framework includes an easy and consistent programming model and a set of APIs, which allow the developer to build applications that uses different programming languages, software, services and devices. Microsoft started the development on the .NET Framework in the late 1990, called originally Next Generation Windows Services. By 2000, first official version of .NET was released (.NET 1.0). Currently, the lasted version is .NET Framework 4.0, released in April 2010 and working only with the new Visual Studio 2010.

Despite .NET Framework is a powerful tool, in the mobile development field a specific version of .NET called .NET Compact Framework is needed. This version is created to provide an environment and access to the underlying features of the mobile devices. It is made to run managed and natives applications concurrently. It also provides interoperability with the Windows CE[3] operating system of a device so the developer is able to access to native functions and integrate native components into the applications. The version used in this report is .NET Compact Framework 3.5, containing a collection of runtime libraries which allow the developer to run applications and services on WM 6.5 devices. The figure 3.2 summarizes the platform architecture and the different development levels.

[3]Windows CE: is an operating system developed by Microsoft for embedded systems.

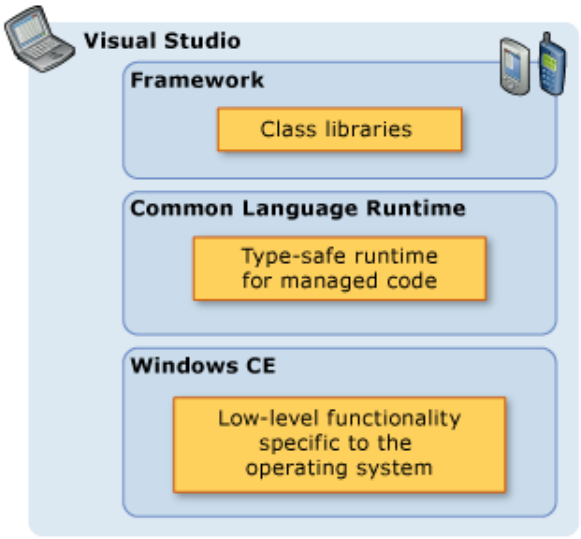

Figure 3.2: System architecture including .NET Compact Framework

3.1.3 C# Programming

With the introduction in the market of the .NET Framework, Microsoft also released a new programming language called C#. C# is an object-oriented language designed to be modern and general-purpose, borrowing concepts from C, C++ and Java with some features of Visual Basic in the mix. C# is theoretically able to be compiled to machine code, but indeed, it is always used with the .NET Framework. Therefore, C# applications require .Net Framework installed on the computer. In C#, everything is wrapped in classes, without offer global variables or functions; even simple types such as string, int, double or char belong to the System.Object class.

The most important features provided by C# are included in the following list:

- Closure functionality using anonymous methods.
- Pointers are missed.
- Automatic memory management and garbage collection are implemented.
- C# supports data encapsulation, inheritance, polymorphism and interfaces. Those concepts bring C# closer to Java language.

- Value types are initialized to zeros and reference types to null by the compiler automatically.
- C# cannot perform unsafe tasks like convert a double to a boolean.
- Partial types allow the separation of a class implementation into more than one source file. This feature allows Visual Studio to generate code, which can be kept separate from developer code (project). This concept notably simplifies the developers task, and it is taken from Visual Basic language.

3.1.4 Terminals Working under Windows Mobile 6.5 Software

At present, there are several terminals running under Windows Mobile 6.5 operative system. WM 6.5 has great support from major mobile phone companies, and a wide list of terminals has been released in the market. The list is complemented with Windows Mobile 6.1 OS terminals that may be upgraded. The most important companies providing Windows Mobile 6.5 terminals are HTC, Samsung, Toshiba, and LG as well as PDAs manufacturers. The largest variety of WM 6.5 devices belongs to HTC, including HTC HD2, HTC Touch 2, HTC Touch Pro 2, HTC Touch Diamond 2, HTC Pure, HTC Imagio etc. Between them, there is also HTC HD Mini, which will be used in this report.

HTC HD Mini is a compact Smartphone (57.7x11.7 mm) with a full touchable capacitive touch screen. The Smartphone allows all necessary functions to interconnect the patient with the hospital in an eHealth Network. The most important mobile specifications, that will be used by AsthmaApp, are described below:

- Platform: Windows Mobile® 6.5 Professional with HTC Sense™.
- Internet: Wi-Fi IEEE 802.11 b/g, 3G up to 7.2 Mbps download speed and up to 2 Mbps upload speed.
- Storage: ROM 512 MB and microSD™ memory card expansion slot.
- Bluetooth®: Bluetooth® 2.1 with Enhanced Data Rate. Widcomm[4] stack implemented.

[4]Widcomm stack: refers to an implementation of the Bluetooth protocol stack.

- Location: internal GPS antenna.
- CPU Processing Speed: 600 MHz.

Figure 3.3: HTC HD Mini

3.2 Getting Started with Windows Mobile Application Development

Amateur and new developers are often intimidated with the troubles surrounding mobile application development. Fortunately, new technologies allow developers and coders to make relatively simple in creating mobile applications. Windows Mobile is an example of simplicity, and in the next section it will be shown how creating and running an application can be surprisingly easy.

3.2.1 Creating, Building and Running an Application

The following steps should be done by the developer in order to create an empty runnable application for Windows Mobile 6.5:

- **Creating a smart device project:** first of all, the user has to open Visual Studio 2008 and to go to File > New > Project. In the templates pane, Smart Device Project has to be chosen. When the Add New Smart Device Project wizard pops up, the developer has to take into account that .NET Compact Framework 3.5 (last version) and Windows Mobile Professional 6 SDK are selected on the top of the window. Then, choosing Device Application and clicking OK button, the application will be created.
- **Adding functionality:** from the Toolbox pane on the left side, many icons can be selected. Dragging icons to the design view, the developer can add functionalities to the application.
- **Building and deploying an application:** the application can be run in a Windows Mobile device or in an emulator provided by Windows Mobile SDK. Once started debug (icon in a toolbar), the developer is able to choose between different kind of emulators and a professional device. Therefore, selecting one of the list and pressing OK, the application should be up and running.

3.2.2 Graphical Interface and Programming Side

Programming with Visual Studio 2008 and Windows Mobile 6 SDK, the user has 2 different frameworks, design and code sides. In the design view is possible to add functionalities to the application, dragging buttons from the toolbox on the left onto the form (device graphical design). Additionally, design view provides a set of properties that can be easily modified such as font, appearance, size and events. A part of code is generated automatically by Visual Studio when properties are changed. The design view also allows the developer to do not waste time programming visual features. It saves time, making the programming experience easier.

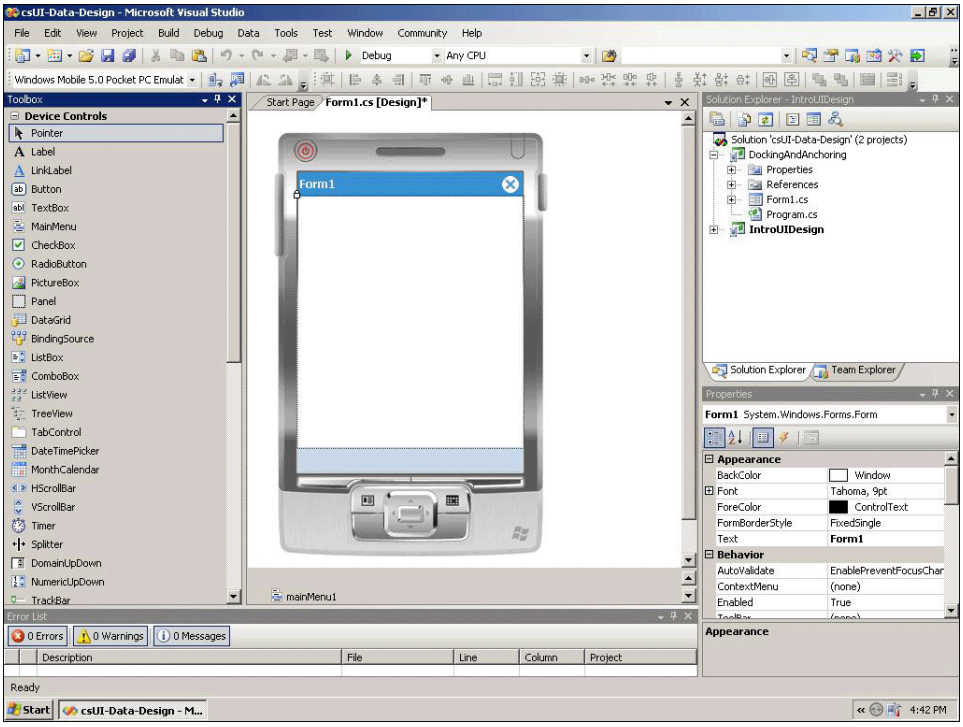

Figure 3.4: Part of design view in Visual Studio for Windows mobile development

On the other hand, the body of the application is situated in the code side (project). In this side, the user is provided by all classes and libraries from .NET Framework and Visual Studio as well as external resources to program the application functions. In addition, it is interesting to mention the autofill option, which makes the programming task easier. In this side, the most important application modules such as bluetooth connection, GPS function, server connection and sms alarms will be programmed .

Chapter 4

Asthma Monitoring Global Network

Asthma Monitoring Global Network is an eHealth system focused on Asthma and COPD sufferers, pretending to be a powerful tool in remote monitoring eHealth. To ensure a successfully remote monitoring, it uses solutions in order to measure all essential necessary parameters to diagnose the disease, including:

- Lung efficiency (spirometry)
- Wheezes monitoring
- Air quality
- Medicine doses and patient symptoms

All those values help to predict possible asthma attacks as well as to value patient's evolution. Therefore, Asthma Global Network contains respectively:

- Medical devices (spirometers)
- Breathing recorders
- Air pollution data base
- Periodical tests

4.1 System Architecture

The Asthma Monitoring System is designed around a central Smartphone, gathering, sending and receiving information from medical devices and external servers. The aim of the architecture design is to provide an easier access to information and services, better patient healthcare services, transparent and efficient use of healthcare resources, and a fast response by the hospital side in case of Asthma attack. The most relevant features and application scenarios will be accurately described in the following sections.

4.1.1 Overall Architecture

The network data flow starts from the side of the user, doing a request and logging into the system. eHealth monitoring system ensures safe and lossless connection between main modules. First of all, the patient starts the daily test answering questions related with breathing and coughing using the Smartphone. Questions differs depending on the test: *morning test*, *evening test* and *on demand test*. The next step is the lung efficiency test. Using a spirometer, the patient takes a PEF/FEV1 test and then, introduces the results into the Smartphone. Finally, by the answers punctuations and taking into account the lung efficiency test, a final health state is set. The health state is displayed in 4 variants depending on the results: *green state*, *yellow state*, *red state* and *panic state*. In the case of panic an alert Sms is sent directly to the doctor and an alarm state to the Hospital Server. After the questions, the patient records the lung sounds using a 4-channel recorder with Bluetooth interface and microphones attached to the stethoscopes. The recorded wav files are sent to the Smartphone via Bluetooth. Finally, test results and wav file reordered are sent to the Hospital Server Data Base and the patient logs out the system. In the Hospital Server side, wav files are analyzed as well as test results. Medical doctors make use of this information and history of previous tests in order to diagnose the patient. Moreover, it exists the possibility of logging into the system only to check the air pollution state. In air pollution requests, GPS antenna is turned on and patient coordinates are sent to the hospital server, which is downloading periodically pollution data. At last, the hospital server sends a response to the Smartphone, which displays the current air pollution state.

On the other hand, medical doctors have an access to the Hospital Server, which allows an easy graphical interface to check patients data history, last tests done and

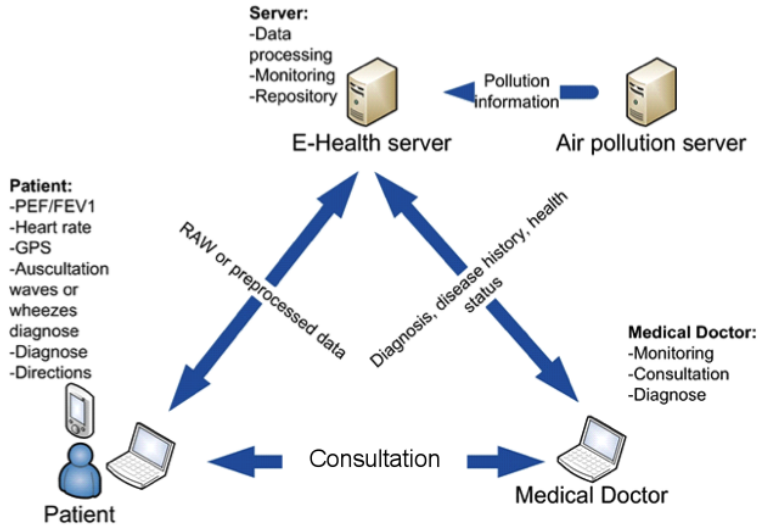

Figure 4.1: Asthma network data flow

wheezes recordings. Medical doctors software provides an easy interface and intuitive design, containing chart visualization, list of patients and other features which make the diagnosis experience easier. Using that powerful tool, doctors are able to diagnose and to treat patients remotely. Consultation between patients and doctors may be performed by using phone, videoconference or in the hospital, depending of the situation and the needs of each case. Figure 4.1 shows Asthma Monitoring Network data flow between the 4 main modules: patient, eHealth Server, doctors and Air Pollution Server.

4.1.2 Networking Aspects

In the Asthma Monitoring Global System, there are several network technologies considered. Each module, depending on its features, implements a technology that fits better to the final block function. The aim of the eHealth System is to provide high reliability as well as QoS (Quality of Service).

The Bluetooth network technology will be used for connections between patient medical devices and the patient terminal (Smartphone), in the frame of Wireless Personal Area Network[D07]. All sensors used in ASTHMA monitoring system are equipped with Bluetooth interface. However, Bluetooth is implemented only in the

breathing recording data reception, considering it is simpler to input PEF and FEV1 values obtained by the spirometer manually to the application. Bluetooth has been designed as an open wireless standard for exchanging data over short distances. It can be used, as in Global Monitoring Asthma Network context, to connect nearby devices and exchange information through a secure radio channel. Despite Bluetooth is a short-range technology, normal ranges are over 70m, enough distance to allow the communication between medical devices and AsthmaApp.

On the other hand, the interaction between the patient's Smartphone and the Hospital Server will be implemented via Wifi or 3G/GSM networks. The network connection is automatically selected depending on the availability of such technology, which is directly related with the patient's location. The connection to the infrastructure network will be switched to Wifi by default, although in a case of unavailability, any other network can be used. Despite in the majority of the scenarios Internet access will be possible, in a case of any network connection available, the patient will be alerted and the information will be saved in the Smartphone's internal memory. Then, AsthmaApp will send the information later. Finally, networking between Hospital Server-Pollution Server-Doctors will be implemented as a normal Internet connections.

At last, as it was aforementioned in Overall Architecture section, communication between medical doctors and patients may be performed via video-conference, telephone or face-to-face consultation, mainly depending on the patient's health state and treatment required.

4.2 Communication Scenarios

In this section, the most important communication scenarios involved in the Asthma Monitoring eHealth System will be described. To ensure a proper monitoring disease control, the system provides 5 different scenarios:

- eHealth server
- Spirometry
- Breathing recordings

- Air quality monitoring
- Consultation

All of them represent communication actions, which are directly interacting with AsthmaApp.

4.2.1 Databases. eHealth Server

Amonit is the Hospital Server, aimed to help monitoring patients suffering for lung diseases – especially asthma and obstructive pulmonary disease. Server application stores the data in a database and provides to physicians a remote monitoring (preventing attacks of the disease). An important goal during creation of server application is to ensure compliance with medical standards: Electronic Health Record (data structure) and Health Level 7 (data exchange). Amonit is provided by:

- **Operating system:** Debian 2.6.26
- **Database language:** MySQL 5.0.51a[1]

The server scripts are written in php5[2] programming language. A huge support community and great number of libraries have been the main reasons in the final choice. Web interface is written using xhtml[3]. The current functionalities of the Hospital Server are described below:

- Receiving and storing measurements and results sent by AsthmaApp.
- Www interface for doctors: notifications about new records in system (with preliminary rating the patient's health risks), presentation entries in tables and chart visualization.
- Sending information about air quality based on GPS coordinates.

[1] MySQL: is a relational database management system hat runs as a server providing multi-user access to a large number of databases.
[2] Php: is a general-purpose scripting language designed for web development.
[3] Xhtml: is an extensible hypertext markup language.

4.2.2 Lung Efficiency Measurements

At the present time lung efficiency measurements are common techniques in Asthma Remote Monitoring Systems. In Asthma Monitoring Global Network, self lung efficiency tests are taken twice a day (morning and evening tests), and the number of tests may be increased sometimes by on demand tests. Self tests using a spirometer medical device are simple and, in addition, they give a precise information about the patient's health state. The spirometer used in the system is a **Spirobank II** from the company Medical International Research. Spirobank II is provided with Bluetooth interface and USB port. Bluetooth communication is performed by serial port profile. Moreover, the device is provided by LCD screen, allowing the patient to visualize all the data taken. Results obtained are reliable because Spirobank II accuracy is ± 5% or 200 mL/s in flow measurements. The following list[Web1] shows the most important spirometer technical specifications:

- Measured values: PEF, FEV1, FVC, FET, VC, IC, ERV, etc.
- Temperature sensor: semiconductor (0-45°C).
- Volume accuracy: ± 3% or 50 mL.
- Flow accuracy: ± 5% or 200 mL/s.
- Display: graphic LCD - FSTN, 128x64 pixel.
- Communication port: USB, Bluetooth®, and RS232.
- Bluetooth profile: service discovery application, generic access, serial port.
- Dimension: 60 x 145 x 30 mm.
- Weight: 180 grams (battery included).

4.2.3 Wheezes Detection

Wheezes detection is a novel signal processing technique in Asthma Monitoring Global Network. Vital signals from the patient are processed by algorithms written in Matlab, in the hospital side. The purpose of mentioned algorithms is to detect wheezes in the reordered breathing sounds. In order to analyze effectively those sounds, recordings are divided in 2 stages. First of all, the patient has to record 3 phases of breathing. Then, a caught is forced to clean up the lungs from secretion. Finally, another 3-breathing cycles are recorded. Following that procedure, clean lungs and lungs with secretion are analyzed. By using those measurements, medical doctors are able to evaluate possible lung obstructions. After caught, patient with well-controlled asthma has usually clearer sounds than COPD patient. This fact is taken into account in the processing algorithms, varying in Asthma and COPD versions. Despite manual methods of lung sounds analysis are widely used nowadays, digital analysis will replace manual methods successfully in the near future. A review of digital wheezes analysis is presented in [WZ10].

Figure 4.2: Wheezes recorder

The heart of wheezes recorder is an ARM[4] processor. All the data are reordered on 16 bits. Additionally, the recorder provides Bluetooth interface and SD card slit to expand internal memory. Bluetooth serial port profile is used in data transmission. The recorder provides 4 channels, each one with microphones attached to the respective stethoscopes. Three microphones take over to get signals from each

[4]ARM processors: are a 32-bit reduced instruction set computer developed by ARM Holdings company.

lung and around trachea, and one microphone records the state of breath (inspiration and expiration). In order to avoid recordings from the background (ambient sounds), lung and trachea microphones are attached to the endings of stethoscopes. Complementing microphones system, an elastic belt is attached to the ending of stereoscopes, protecting and ensuring uncorrupted recordings. The main objective of this device is to record the lung sounds as clean as possible in the breathing cycles.

Taking into consideration that the wheezes recorder was developed and made by AGH university, one of the most important drawbacks of the system is that currently there is only one recorder available. In the next months a set of similar recorders will be bought in order to cover the market demands.

4.2.4 Air Quality Monitoring

Another relevant aspect in remote monitoring asthma systems is the air pollution control. The growth of industrialization and urbanization has increased the air pollution concentration. It becomes more and more serious in any part of the world. Asthma patients react on significant changes of temperature and humidity, especially on temperature decreases. Furthermore, it is also highly recommended to asthma sufferers avoiding air with high pollution levels. AsthmaApp, by using the Smartphone GPS antenna, informs to the patient about air quality state. To achieve the current air pollution state, Internet access and GPS range must be available, otherwise the petition will not be served. In the Asthma Monitoring Global Network, air pollution service is implemented as an On Demand request, checking the data only under patient's petition. This system is useful in the case of patient breathing deterioration, checking in real-time whether the discomfort may be directly caused by air pollution. In figure 4.3 it is shown the air quality network service.

It is also important to notice that the Hospital Server is updating real-time data from external servers every hour, storing air pollution quality in its databases. Hence, the patient is able to download the current pollution state as many times as he/she needs during the day. Professional services monitor wide range of pollutions, which may be dangerous to an asthma patient such as SO2, NO, NO2, NOx, COx, O3, PM10[5] or PM2.5[6]. The current information source is the website of Małopolski Monitoring Powietrza. It provides data gathered from 11 monitoring stations spaced in province

[5]PM10: microscopic dusts with diameter less than 10 micrometers.
[6]PM2.5: microscopic dusts with diameter less than 2.5 micrometers.

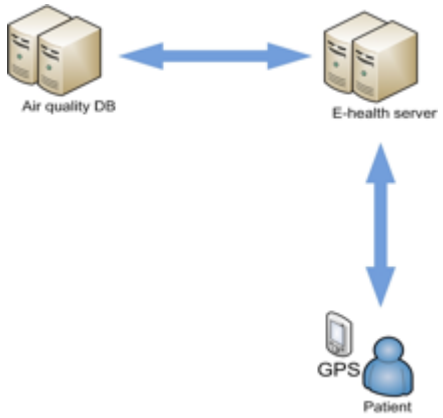

Figure 4.3: Air Pollution Network

Małopolska (Poland). The criterion of patient's health risk is the content of nitric oxide in the air. Depending of the following nitric oxide levels, a pollution state is set:

- Green: (0, 40)μg/m3
- Yellow: (40, 400)μg/m3
- Red: (400, 500)μg/m3
- Panic: above 500μg/m3

4.2.5 Consultation

AsthmaApp provides valuable information about patient vital signs, necessary to monitor and to diagnose asthma disease. After being processed, the information about each patient is displayed into a simple an easy environment, in order to group the data in the best way, making the doctor's diagnosis easier. Depending on situation, doctor availability and patient preferences, each consultation may be performed by voice representation, high quality video streaming (videoconference) or normal face-to-face consultation in the hospital. Additionally, an alarm system is implemented, consisting of sms message, which allows an instantaneous notification of the alarm state. Then, a direct interaction between patient and doctor may be possible in case of asthma attack risk.

4.3 Data Specification

Global Network manages different types of data in two-way communication. As it was shown in figure 4.1, all data in the disease remote monitoring go through the Hospital Server. Input data are defined as incoming data to the Hospital Server (eg. data transmitted from the Smartphone, patient data). Consequently, outcoming data from the Hospital Server are considered output data (eg. data taken by the medical doctors).

4.3.1 Input Data

At the end of each Daily Test, AsthmaApp has the necessary information to diagnose the disease remotely. Then, data obtained by medical devices and questions are sent to the Hospital Server, as an input data. In table 4.1 all different kind of data and frequency per day are shown. On demand requests frequencies are defined by X, where X means an indeterminate number of times.

Input data	**Frequency**	**Data description**	**From**
Patient information	First time	Name, surname, gender, age, height	Medical Doctor
Daily Test results	2/24h	Test marks (morning and evening)	AsthmaApp
On Demand tests	X/24h	Test mark	AsthmaApp
PEF	2/24h	Value (spirometry)	AsthmaApp
FEV1	2/24h	Value (spirometry)	AsthmaApp
Wheezes	2/24h	.wav file (recording)	AsthmaApp
GPS	X/24h	Longitude, latitude	AsthmaApp
GPS	24/24h	SO2, NO, NO2, NOx, CO, O3	Air Pollution Server
Alarms	X/24	Panic state	AsthmaApp

Table 4.1: Input data stored in Medical Data Bases

4.3.2 Output Data

On the basis of input data, doctors are able to diagnose the patient. The output flow data mainly happens on the Medical Doctor - Hospital Server communication way. The following table shows the whole output data flow specifications and destinations:

Input data	Data description	To
Daily Test results	Test marks (charts)	Medical Doctor
On Demand tests	Test mark (charts)	Medical Doctor
PEF	Value (charts)	Medical Doctor
FEV1	Value (charts)	Medical Doctor
Wheezes	.wav file (recording)	Medical Doctor
GPS	Longitude, latitude	AsthmaApp
Alarms	Panic state	Medical Doctor

Table 4.2: Output data in Medical Data Bases

Patient information is displayed using charts to the doctors, providing an easy way to interpret the results in order to make a diagnose as well as possible. Additionally, it is possible to listen breathing recordings from the doctors side. Finally, Asthma Monitoring System allows an instant Sms alert to the doctor's mobile phone in a case of asthma attack risk.

Chapter 5

AsthmaApp

In this chapter the main functions and features of AsthmaApp will be described. AsthmaApp is aimed to create an easy and intuitive environment, providing the necessary keys to monitor Asthma and COPD diseases. On the basis of C# programming, different forms have been created, interacting between them in order to have an access to the different application functions and resources. Hence, the user can surf trough the forms changing applications settings or the patient personal information, consulting last tests results or air quality state as well as eHealth Monitoring tasks, such as Daily Tests. Consequently, the application has been thought to provide a powerful self-care tool, which can be used several times during the day, not only in the context of eHealth Monitoring.

In the next sections, the reader will get straight to the point of the most important application modules, programming issues and necessary aspects to obtain a well performance and behavior of the application. Furthermore, medical specifications and algorithms to set health states will be accurately described.

5.1 AsthmaApp Architecture

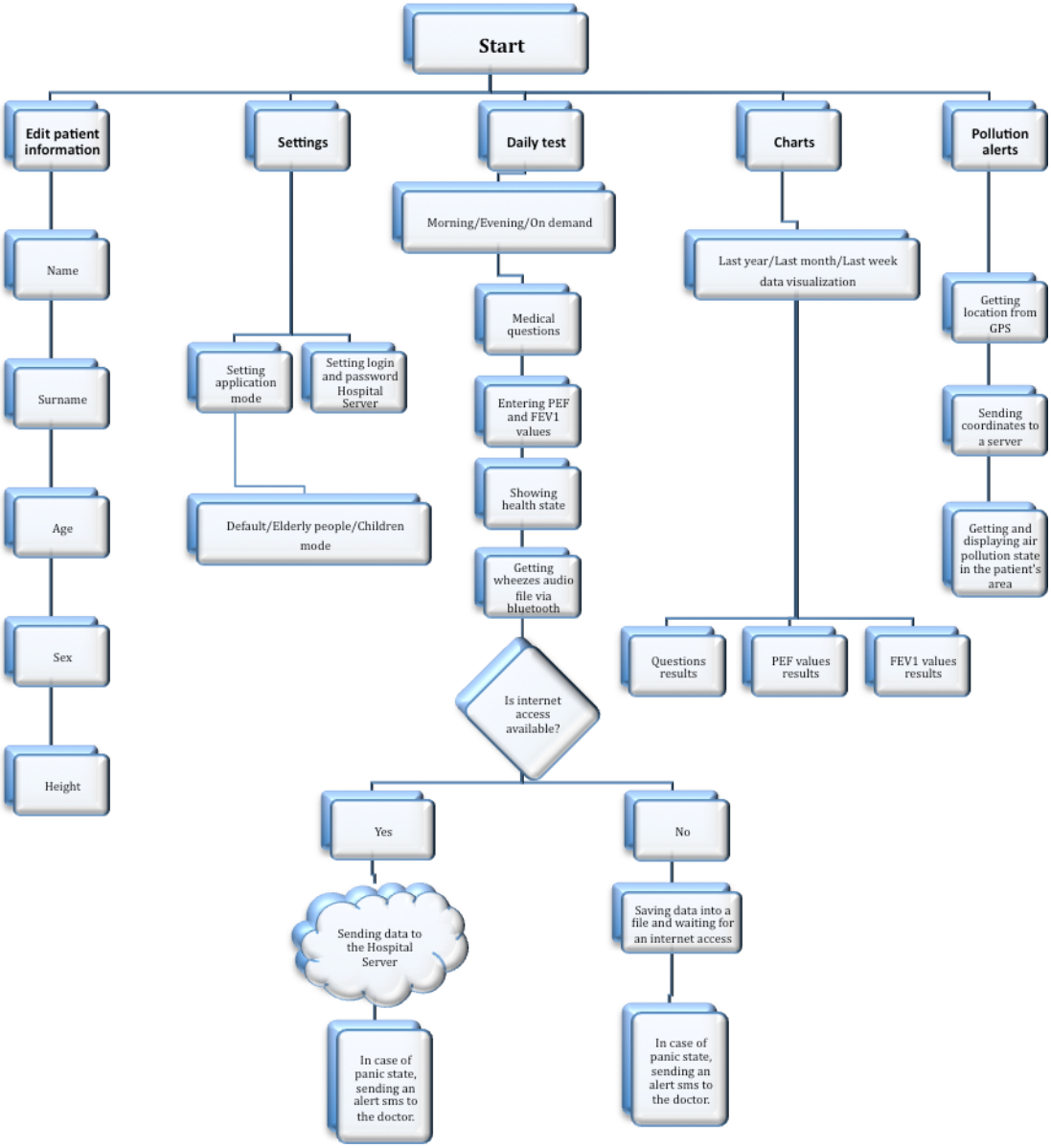

Figure 5.1: AsthmaApp architecture

Figure 5.1 shows the whole functionalities of AsthmaApp, explaining which steps the application is following in the possible scenarios. It is clear that AsthmaApp can be divided in 5 modules: edit patient information, settings, daily tests, charts and pollution alerts. Each one deals with different functions, all of them necessaries to obtain the final result. As it is shown in the flow diagram, the most complex branch, which includes more eHealth techniques, is the Daily Test algorithm.

5.2 Medical Requirements

Medical doctors have had a relevant importance in this project. Their guidelines and requirements have been implemented in AsthmaApp in order to ensure a proper disease monitoring. Daily Test questions were chosen by Jana Pawła Hospital lung experts, who have an experience and wide knowledge of Asthma and COPD diseases. PEF and FEV1 measurements are compared with well-known standards values, depending on the gender, age and height of the patient. Finally, in the case of wheezes recordings, there is no directive to follow, assuming that the analysis of the data will be done in the Hospital Server side.

5.2.1 Daily Test Questions

Morning, evening and on demand tests are composed of different questions and number of inhalations (times that the patient takes medicine). Then, morning tests have 1 question whereas evening and on demand tests have 2. Questions are related with the health experience of the patient last hours. On the other hand, the number of inhalations is taken in order to corroborate the authenticity of the answers, taking into account that the patient feelings are always subjective. Each question has 5 possible answers, according with the patient health state. Answers a) are always referred to the best health state whereas answers d) are referred to the worst. In addition, the questions are also plotted in different colors (green, yellow, red, dark brown and black respectively) according to the patient's condition. It is thought to help answering in a visual way. Finally, a punctuation is given according to the answer, as it will be described below.

5.2.1.1 Morning Questions

How did You sleep last night?

a) I didn't have any problems at night (4 pts)

b) Because of the symptoms, I woke up once or earlier (3 pts)

c) Because of the symptoms, I woke up more than once (2 pts)

d) Because of the symptoms, I woke up for majority of the night (1 pts)

e) The symptoms were so annoying that I didn't sleep at all (0 pts)

5.2.1.2 Evening Questions

Did you have any difficulties in breathing today?

a) None (4 pts)

b) A few, while making big effort (e.g. running) (3 pts)

c) Medium, while making little effort (e.g. making bed) (2 pts)

d) Significant, while making everyday activities (e.g. getting dressed) (1 pts)

e) Serious, almost continuous, even while having a rest (0 pts)

How often did you cough today?

a) No cough, I didn't cough at all today (4 pts)

b) Rare, once in a while (3 pts)

c) From time to time, less than once an hour (2 pts)

d) Often, more than once an hour (1 pts)

e) Almost continuous, I was coughing or going to cough all the time (0 pts)

5.2.1.3 On Demand Questions

Because of the fact that the "on demand" test is done when the patient feels discomfort, the green gate is cancelled. Only 3 answers are available.

Did you have any difficulties in breathing recently?

c) Medium, while making little effort (e.g. making bed) (2 pts)

d) Significant, while making everyday activities (e.g. getting dressed) (1 pts)

e) Serious, almost continuous, even while having a rest (0 pts)

How often did you cough recently?

c) From time to time, less than once an hour (2 pts)

d) Often, more than once an hour (1 pts)

e) Almost continuous, I was coughing or going to cough all the time (0 pts)

5.2.1.4 Questions Punctuation and Correlation with Number of Inhalations

As it is shown in the last subsections, a number of points is given to each answer, corresponding 4 points to answer a) (green state), 3 points to answer b) (green state), 2 points to answer c) (yellow state), 1 point to d) (red state) and 0 points to e) (panic state). To ensure reliable results, a correlation is done always between dyspnoea[1] questions and the number of inhalations. Dyspnoea answers should be around or equal with number of inhalations:

- Answer a) $\leftrightarrow$ 0 inhalations (green)
- Answer b) $\leftrightarrow$ 1 inhalation (green)
- Answer c) $\leftrightarrow$ 2 inhalations (yellow)

[1] Dyspnoea: difficulty in breathing.

- Answer d) ↔ 3 inhalations (red)
- Answer e) ↔ 4 inhalations (panic)

If the difference between answers and inhalations is greater or equal than 2, points are added or subtracted to the answer punctuations depending of the situation. Concretely, whether the number of inhalations is greater or equal than 2, the dyspnoea answer punctuation is subtracted as many times as the number of inhalations minus the answer punctuation plus 1; up to 0. In the case of the number of inhalations is lesser or equal than 2, the opposite algorithm is done. In conclusion, the differences minus 1 are added or subtracted to the final result, making the answers more reliable. The following examples make the correlation process clearer:

- Answer a) and Inhalations = 2: difference = 2, then the final question punctuation corresponds to b).
- Answer c) and Inhalations = 3: difference = 0, any change.
- Answer e) and Inhalations = 0: difference = 4, then the final question punctuation corresponds to b).

Furthermore, in evening and on demand tests, questions about dyspnoea have higher priority than questions about cough. It is assumed that the ratio between dyspnoea questions and cough questions is about 2:1. Hence, points from dyspnoea are multiplied by 2. Finally, evening and on demand tests have a punctuation from 0 to 12 (4 + 4*2) whereas morning tests have a punctuation from 0 to 4. Therefore, a partial state is set depending of the answers punctuation. The following list shows an approximation between possible results in morning and evening/on demand tests, and their parallel health state.

Morning tests

- 3-4 points: green
- 2 points: yellow
- 1 points: red
- 0 points: panic

Evening/on demand tests:

- 8-12 points: green
- 6-7 points: yellow
- 4-5 points: red
- 0-3 points: panic

It is important to take into account that those results are not the final patient health state. They only correspond to the questions block.

5.2.2 PEF/FEV1 Values

The next step after answering the questions is the spirometry. The patient has to take a lung test using an spirometer and then, introduce PEF and FEV1 values into the Smartphone. Once introduced those data, the results are compared with standard predictions mentioned in the tables 2.1, 2.2, 2.3 and 2.4. Results obtained, depending of the percentage of standard values, receive a punctuation as well as in the questions block. Punctuations and percents of standard values are summarized below:

Morning tests

Percentages of standard values (tables*0.8):

- 100% - 85% ↔ 4 points (green state)
- 85% - 70% ↔ 3 points (green state)
- 70% - 60% ↔ 2 points (yellow state)
- 60% - 40% ↔ 1 point (red state)
- < 40% ↔ 0 points (panic state)

Evening/on demand tests

Percentages of standard values:

- 100% - 93% ↔ 12 points (green state)
- 93% - 85% ↔ 11 points (green state)
- 85% - 78% ↔ 10 points (green state)
- 78% - 70% ↔ 9 points (green state)
- 70% - 65% ↔ 8 points (yellow state)
- 65% - 60% ↔ 7 points (yellow state)
- 60% - 50% ↔ 6 points (red state)
- 50% - 40% ↔ 5 points (red state)
- 40% - 35% ↔ 4 points (panic state)
- 35% - 30% ↔ 3 points (panic state)
- 30% - 25% ↔ 2 points (panic state)
- 25% - 20% ↔ 1 point (panic state)
- < 20% ↔ 0 points (panic state)

Notice that at morning, PEF and FEV1 values are usually 20% lower than standard values. This key is taken into account in order to set the final lung test gate. After this process, lung efficiency values and questions results have the same standard punctuation.

5.2.3 Setting a Final Health State

The last step is to set a final health state, which will be displayed to the patient. Lung efficiency measurement have higher priority than the questions (60% against 40%) because of the fact that answering questions is not an objective task. In the frame of lung efficiency measurements, the ratio between PEF and FEV1 values is 50% - 50%. Finally, the final state can be easily calculated by using a simple equation:

$$FinalState_{Points} = Questions_{Points} * 0.4 + (PEF_{Points} * 0.5 + FEV1_{Points} * 0.5) * 0.6$$

At last, depending on the points obtained, a proper final health state is set, as it is described below:

Morning tests

- 3-4 points: green state
- 2 points: yellow state
- 1 points: red state
- 0 points: panic state

Evening/on demand tests:

- 8-12 points: green state
- 6-7 points: yellow state
- 4-5 points: red state
- 0-3 points: panic state

5.3 Graphic Design and Forms

In this section few aspects concerning the visual design of the application as well as the structure will be discussed.

5.3.1 Graphic Design

Creativity and technical ability are important to communicate with visuals. It is needed attentiveness to detail and an ability to express broad, complex ideas and emotions with commonly understood symbols, colors, themes, and characters. Graphic design in AsthmaApp is aimed to transmit the information in an easy, clear, and intuitive way. On the basis of AsthmaApp will be run in a full touchable mobiles, bottoms, textboxes and other objects have been designed in order to be accessible. They are made in appropriate scale, ensuring enough size to avoid possible mistakes in touching.

On the other hand, bluish and white colors have been selected to the background, combined with intense colors such as red and green to the other components. Buttoms to access to the different forms have been designed with a white background color and dark purple font. Therefore, colors make the visualization easier. In the test answers, colors are related with health states (e.g. green: wellness or red/black: asthma attack risk). Figure 5.2 shows the visual design of the main form.

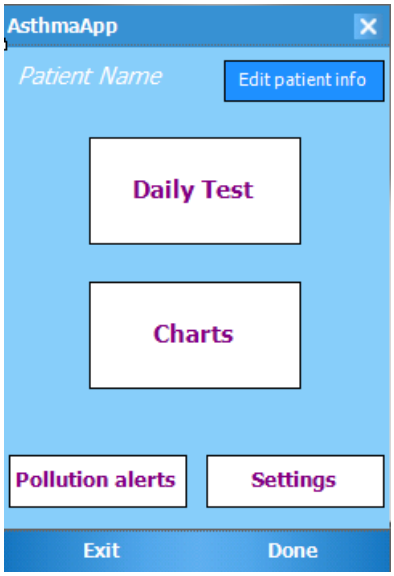

Figure 5.2: Main form visual design

5.3.2 Forms Management

The application is composed by different forms. The user is able to surf trough each application functions by opening and closing forms. The way to move between forms is implemented by buttons. Each time that a bottom is clicked, a part of code is run; creating, showing, hiding or closing forms. When the programs starts, the main form is opened, which will be all the time active. From this form, all the options are available. To create and show a form in C#, it is necessary the following code:

Algorithm 5.1 Creating and closing forms

```
Form1 form = new Form1();
form.ShowDialog();//ShowDialog stops the execution of the current form code, starting with the new form code
form.Close();
```

At last, closing forms is an easy task, and it can be done by the last instruction above.

5.3.3 AsthmaApp Components

In this section, the most important C# classes used as primary components will be described. They may be created in the code side, or just added dragging the icon to the design view.

- **TextBox:** represents a text control. The user may input text and numbers as a string.
- **Button:** can be clicked by using the finger and executes a separate part of code. It is used to open/close forms, accept instructions, etc..
- **Label:** label controls are typically used to provide descriptive text for a control.
- **LinkLabel:** is a label control that can display hyperlinks.
- **ListBox:** enables you to display a list of items to the user that can be selected by clicking.
- **ComboBox:** displays a text box combined with a ListBox, which enables the user to select items from the list or to enter a new value.

5.3.4 First Run and Welcome Form

The first time the program runs in a mobile device, a welcome registration form is enabled. The application creates automatically a new directory and a file with the patient personal information introduced. Next times, when the application runs, the first programmed step is to check whether the Asthma application directory exists. Then, in the case of existence, another version of welcome form (without registration step) it is shown. By using the mentioned algorithm, AsthmaApp is able to distinguish the first run, obtaining data needed in future remote monitoring tasks. In order to create a directory and a file, the code below is required:

Algorithm 5.2 Creating directory and application data files

```
//Creating a directory in Application Data Folder from the device
System.IO.Directory.CreateDirectory(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
+ "\\AsthmaAppData");
//Creating a called Mode.txt
System.IO.FileStream sw = new System.IO.FileStream(Environment.GetFolderPath
(Environment.SpecialFolder.ApplicationData) + "\\AsthmaAppData\\Mode.txt", System.IO.FileMode.OpenOrCreate,
System.IO.FileAccess.Write, System.IO.FileShare.Read);
//StreamWriter class allows to write chars into a files
System.IO.StreamWriter swrite = new System.IO.StreamWriter(sw);
```

5.3.5 Application Modes

AsthmaApp is thought to cover the totality of asthma sufferers. Hence, 3 different application modes have been programmed: default, children and elderly people. The differences between modes can be appreciated only in daily test section. Default mode presents standard questions, size and colors whereas children version is provided with shorter questions and bigger fonts, in order to make the test more understandable. Finally, elderly people mode is focused on visualization, trying to provide a big font and colorful questions, aimed to patients with short-sightedness problems. Figure 5.3 shows the differences between application modes in the same daily test question.

By using a variable saved into a file, the application is checking in every run the mode selected, switching automatically the correct form. Furthermore, if the user changes the application mode in the settings section, AsthmaApp memorizes the mode selected overwriting the new value into the internal file.

Figure 5.3: AsthmaApp modes (default on right, children on center and elderly people on left)

5.4 AsthmaApp Data

AsthmaApp deals with 2 important data types. The application uses data files saved in the internal Smartphone memory. Those files provide all the information related to the patient, such as tests results, patient personal information or Hospital Server login and password. A good management of the internal files is essential to ensure a proper whole system performance. Secondly, the other type of data involved in AsthmaApp is the data sent to the server. The Hospital Server has an algorithm to difference all the requests and to serve them separately. In order to ensure a proper recognition, each data type is preceded by one identifier: 1, 2, 3, 4 or 5. In the following section data structures will be discussed.

5.4.1 Data Structure

The data sent to the server is divided in 5 types of primitives, 4 of them related to tests results and requests, sent as strings, and the last is related to .wav files, sent as a bit array.

5.4.1.1 Results Sent to the Server

There are implemented 4 kinds of request in a string format. The values sent are separated by "#" and they start with an identifier. At last, 4 "#" are added to the end of the string. According to the structure mentioned, the different requests have the following format:

1. **Morning test results**: identifier= 1 **Format:** 1#question_value#PEF_value#FEV1 value####
2. **On demand test results**: identifier= 2 **Format:** 2#q1_val#q2_val#PEF_val#FEV1 val####
3. **Evening test results**: identifier= 3 **Format:** 3#q1_val#q2_val#PEF_val#FEV1 val####
4. **Air quality request**: identifier= 4 **Format:** 4#latitude#longitude####

To ensure a proper detection from the Hospital Server side, the data shown above is included in a main string, which will be sent encoded UTF8[2]. Finally, the string transmitted is:

- data='*data string(1,2,3 or 4)*'&op=2&user='*user login*'

The strings are uploaded by using the class *WebResponse* from .Net Framework. Programming detalis concerning the C# code will be discussed in the Connectivity section.

[2]UTF8: multibyte character encoding for Unicode.

5.4.1.2 .wav Files Format

Uploading a binary file to an external server is not an easy task in C#. Several headers and control values are required before and after to send the own file. The .wav file is also uploaded as HTTP post by using the class *WebRequest*. This requirement has been one of the most difficult tasks. Working close with the Hospital Server side, a file was uploaded via website. Then, by means of a HTTP analyzer program, traffic was captured in real-time. Once checked the header, content, cookies, query strings, post data, request and response streams, the necessary information to upload a file was discovered. Finally, a proper header has been created, made with the description of the file and previous required information. A long string is sent as a header, containing the data below:

```
-----------------------------41184676334//Boundary
Content-Disposition: form-data; name="MAX_FILE_SIZE"

50000000
-----------------------------41184676334//Boundary
Content-Disposition: form-data; name="oopp"

251
-----------------------------41184676334//Boundary
Content-Disposition: form-data; name="userfile"; filename="File.wav"
Content-Type: audio/wav
```

Following the header, the binary file is sent. Finally, an ending is also required to finish the Post request:

```
-----------------------------41184676334//Boundary
```

Moreover, AsthmaApp is analyzing the response from the Hospital Server and informing to the patient whether the file has been uploaded successfully.

5.4.2 Application Files

The application files created after first run, such as test and patient data files, are located in AsthmaApp data folder whereas the files containing PEF and FEV1 values remain in the application directory. In the AsthmaApp data directory 4 files have been created: datapatient, mode, morning test and evening test. Data patient file

as well as Mode file are created in the first run. Mode file is composed by 3 lines: application mode, ans Hospital Server patient login and password. On the other hand, Data patient file contains 5 lines with name, surname, gender, age and height respectively. In the following subsections the content and structure of the other files will be described.

5.4.2.1 Morning/Evening Files

First of all, it is important to mention that AsthmaApp is able to memorize 1 year of daily test results. All the information concerning the test, PEF and FEV1 results may be always checked in the *Charts* section. Because of the fact that AsthmaApp is writing into the files using the class *StreamWriter*, and the method *WriteLine*[3], all the lines with data must have the same length. Otherwise in each writing exists the possibility of keep useless data or even delete useful data after each "\n". Therefore, morning and evening test files are composed of the date of last test in the first line, and 365 lines of 20 characters. Each line contains test results, PEF values, FEV1 values and a set of zeros filling the rest of line. Those values are separated by ";". Moreover, the date of last test is a number expressed as a day of the year, being in a range between [1-366]. AsthmaApp takes the day of the year from the class *DateTime* and method *DayOfYear*. In addition, the files are initialized with the current day of the year minus 1 in the first run, and with '-1' in the place of tests values. When a daily test is missed, the files are uploaded writing a line with zeros in the proper day position. A flow diagram with the accessing file algorithm is described in figure 5.4. An example of morning/evening test file is also shown below:

```
6
-1;-1;-1;0000000000;
-1;-1;-1;0000000000;//Days without test
10;789;2.67;000000;
9;656;1.9;00000000;
0;0;0;000000000000;//Test missed
8;599;1.232;000000;//Last test
...
-1;-1;-1;0000000000;
```

The example shows different possibilities: normal tests, empty days and daily test missed days.

[3]WriteLine: writes a specified string followed of "\n".

5.4.2.2 PEV/FEV1 Files

PEF and FEV1 files provide the standard values necessaries to compare with spirometry results in each daily test. Values are separated by ";". In order to have a correct access to the file in the application, it is important to set in Visual Studio the property *"copy into the directory"* to *"copy always"*, otherwise the file will not be copied and consequently, inaccessible.

5.5 Connectivity

This section will go to the point in programming issues concerning communication modules available in AsthmaApp. Internet access and bluetooth connection are the most important modules to perform a disease remote monitoring. However, Gps connection makes easier the disease prevention, allowing the patient to avoid polluted areas. Finally, sms alarms may be essential in the case of asthma attack.

5.5.1 Internet Connection

The server connection is the most important connectivity module, allowing the remote disease monitoring. In order to create a Http post in C#, *WebRequest* class is used. By using the System.Net assembly, *WebRequest* class becomes available. First of all, the *WebRequest* properties must be set (method to "POST", content type to *"application/x-www-form-urlencoded"* and content length to the length of the data string). Then, a string with useful data is UTF8 encoded. After that, the request to the Hospital Sever is sent by calling the *GetRequestStream* method, and a network stream is gotten. Finally, the connection is established with the server, and writing to the stream that holds request data, the information is uploaded. At last, the developer may call the method *GetResponse* in order to read the response from the server. *GetResponse* shows information about connection such as whether the request has been served successfully. The following example presents a reduced and non error-controlled code that illustrates the most important steps in an Http post. All steps are commented for a better compression.

Algorithm 5.3 Server connection

```
// CONNECTION TO A SERVER
        // Create a request to a Hospital Server URL that can receive a post.
        WebRequest request = WebRequest.Create("http://amonit.kt.agh.edu.pl/astma/index.php");
        // Set the Method property of the request to POST.
        request.Method = "POST";
        // Create POST data and convert it to a byte array.
        //Set the string that will be sent to the server
        string postData = "Useful data from AsthmaApp";
        byte[] byteArray = Encoding.UTF8.GetBytes(postData);
        // Set the ContentType property of the WebRequest.
        request.ContentType = "application/x-www-form-urlencoded";
        // Set the ContentLength property of the WebRequest.
        request.ContentLength = byteArray.Length;
        // Get the request stream.
        dataStream = request.GetRequestStream(); //dataStream is a Stream variable
        // Write the data to the request stream.
        dataStream.Write(byteArray, 0, byteArray.Length);
        // Close the Stream object.
        dataStream.Close();
        // Get the response.
        WebResponse response = request.GetResponse();
        StreamReader reader = new StreamReader(response.GetResponseStream(), Encoding.UTF8);
        // Read the content.
        string responseFromServer = reader.ReadToEnd();
        reader.Close();//Closing objects
        response.Close();
```

5.5.2 Bluetooth Connection

Bluetooth connection has been another difficult task to implement in AsthmaApp. HTC HD Mini is provided by *Widcomm Stack*. *Widcomm Stack* is used by new devices with Windows Mobile 6.5 or Windows Phone 7 OS because it is a modern, well-programmed and efficient stack that supports several bluetooth profiles. Previously, *Windows Bluetooth Stack* was widely used as a software dealing with bluetooth hardware. The main problem is that Windows do not support *Widcomm Stack* (they do not have libraries) and a third party library has been used to implement the bluetooth module. *32feet.Net* is a shared-source project made by *In The Hand*[Web3] company that makes personal area networking technologies such as bluetooth more accessible using .Net code. They have bluetooth libraries supporting *Microsoft Bluetooth Stack* and *Widcomm Stack* as well. The main benefit of using this third party library is that AsthmaApp is able to run with both bluetooth stacks. In order to ensure

a proper *Widcomm Stack* support, "InTheHand.Net.Personal.dll" must be added in a references folder from Visual Studio and the native mapping DLL “32feetWidcomm.dll” must be in the same folder that AsthmaApp.exe. Finally, the developer is able to start programming a bluetooth connection.

The steps followed by AsthmaApp to connect the application with the recorder are described in the list below:

- **Turning on the radio in a discoverable mode:** this step allows the recorder to detect the Smartphone as well as the Smartphone collects information about surrounding discoverable and paired devices.
- **Ensuring discoverable mode:** it is important to check whether the bluetooth radio is discoverable, otherwise the recorder will not detect the mobile and the data transmission will not be possible.
- **Selecting a device with a ComboBox:** the information obtained is stored into ComboBox Data Source. Then, the patient can select a device from the list to start the connection.
- **Starting connection:** once a bluetooth device is selected, AsthmaApp starts the connection and gets the stream. From that stream the .wav file will be read.
- **Reading data:** finally an algorithm is programmed to read the full .wav file.

As in the server connection case, the following code gives a simplified example of the classes and methods involved in bluetooth connection. The code is reduced and non error-controlled, just illustrating remarkable aspects of the connection setting up.

Algorithm 5.4 Bluetooth connection

```
private void button5_Click(object sender, EventArgs e)
    {
        cli = new BluetoothClient();
        memstream = new MemoryStream();
        radio = BluetoothRadio.PrimaryRadio;//Selecting bluetooth radio
        if (radio == null)
        {//In case of incompatibility
            MessageBox.Show("No support Bluetooth radio/stack found");
        }
        else (radio.Mode != InTheHand.Net.Bluetooth.RadioMode.Discoverable)
             radio.Mode = RadioMode.Discoverable;//Ensuring discoverable mode
        BluetoothDeviceInfo[] info = cli.DiscoverDevices(255, true, false, true); //Paired and discoverable devices
        comboBox2.DataSource = info;//Assigning the data to combobox
        comboBox2.DisplayMember = "DeviceName";
    }
private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
    {//Selecting a device in the list
        device = comboBox2.SelectedItem as BluetoothDeviceInfo;
    }
 private void button6_Click(object sender, EventArgs e)
    {//Establishing communication
        BluetoothAddress addr = device.DeviceAddress;
        BluetoothEndPoint ep = new BluetoothEndPoint(addr, BluetoothService.SerialPort);//Setting bluetooth protocol
        cli.Connect(ep);
        peerStream = cli.GetStream();//Getting the string, a bidirectional communication is allowed
        readData();//Then, the part of code for read the data in executed
    }
```

5.5.3 GPS

GPS connection allows to obtain in a real-time the longitude and latitude of the current patient's location. Through the activation of the built-in GPS antenna from the Smartphone, AsthmaApp has an accesses to the mentioned coordinates as well as the patient's current speed. For a proper use of that technology, the patient must be located in an open area, otherwise it is impossible to be in the satellites range.

To perform a good management of the GPS hardware, AsthmaApp attaches as a reference a GPS sample project made by Microsoft. By using this sample project, AsthmaApp has an access to classes such as *Gps* and *GpsPosition*, which allow an easier GPS management. Firstly, it is important to create event handlers in order to detect every change from Gps data (longitude, latitude or speed). Then a part of code is run, getting and saving the data into strings. Finally, the information is displayed

and sent to the server. The following code shows the most important steps to get data from Gps. The method *Invoke();* executes the specified delegate[4] on the thread that owns the control's underlying window handle. It is necessary to deal with the GPS sample code provided by Microsoft.

Algorithm 5.5 Getting Gps data

```
private Gps gps = new Gps();
private GpsPosition pos = null;
private void InitGps()
    {//Handlers necessaries to get the data
       saveLocationHandler = new EventHandler(SavePosition);
       gps.LocationChanged += new LocationChangedEventHandler(gps_LocationChanged);
    }
protected void gps_LocationChanged(object sender, LocationChangedEventArgs args)
    {
       pos = args.Position;//We are using samples.location project to get position
       Invoke(saveLocationHandler);//Now we give back the focus to AsthmaApp thread
    }
 private void SavePosition(object sender, System.EventArgs args)
    {
       if (pos != null)
       {
          if (pos.LatitudeValid)
          {
             tim.Enabled = false;
             latitude = Convert.ToString(pos.LatitudeInDegreesMinutesSeconds);
             if (latitude.Length > 14)
             {
                latitude = latitude.Substring(0, 14);
             }
             lat = Convert.ToString(pos.Latitude);
          }
          if (pos.LongitudeValid)
          {
             tim.Enabled = false;
             longitude = Convert.ToString(pos.LongitudeInDegreesMinutesSeconds);
             if (longitude.Length > 14)
             {
                longitude = longitude.Substring(0, 14);
             }
             lon = Convert.ToString(pos.Longitude);
          }
       }
    }
```

[4]Delegates: are similar to function pointers in C or C++ languages.

5.5.4 SMS Alerts

Windows Mobile 6 SDK provides a namespace called *Microsoft.WindowsMobile.PocketOutlook*. This assembly allows the developer to create and access PIM data items (Appointments, Tasks, and Contacts), and MAPI messaging items (e-mail and SMS messages). AsthmaApp uses the class *SmsMessage* in order to send a sms alert to the doctor's mobile in case of panic state (risk of asthma attack). After creating a *SmsMessage* object, the final required steps to send a sms are fill the body (text), add a number. Then, calling the method Send() the operation is completed. The error-controlled code below illustrates how to send a sms:

Algorithm 5.6 Sending sms

```
using Microsoft.WindowsMobile.PocketOutlook;

try//Sending sms alert
    {
        SmsMessage sms = new SmsMessage();//Creating a new SmsMessage object
        sms.Body = "The patient"+login+"has an asthma attack risk!!";//Message text
        sms.To.Add(new Recipient("783614210"));//Here it should be filled with the doctor's mobile
        sms.Send();
        MessageBox.Show("Alarm sent to the doctor");
    }
    catch (Exception)//Error control
    {
        MessageBox.Show("Error sending sms alert");
    }
```

5.6 Daily Test Forms

Daily test forms are the most important forms in the application. They are helping the patient to take the tests and to ensure a proper asthma remote monitoring. The next sections will describe the algorithms used as well as the steps required to take daily tests successfully.

5.6.1 Daily Tests Algorithms

In order to ensure a correct usage of daily test, an algorithm has been programmed. The aim of this algorithm is try to avoid possible misuses of daily test. It is essential

to take into consideration that daily tests must be taken once per day and, in a case of omission, internal files must be updated. According to those directives, before starting daily tests, a set of verifications is done. Figure 5.4 shows a flow diagram explaining the implemented algorithm:

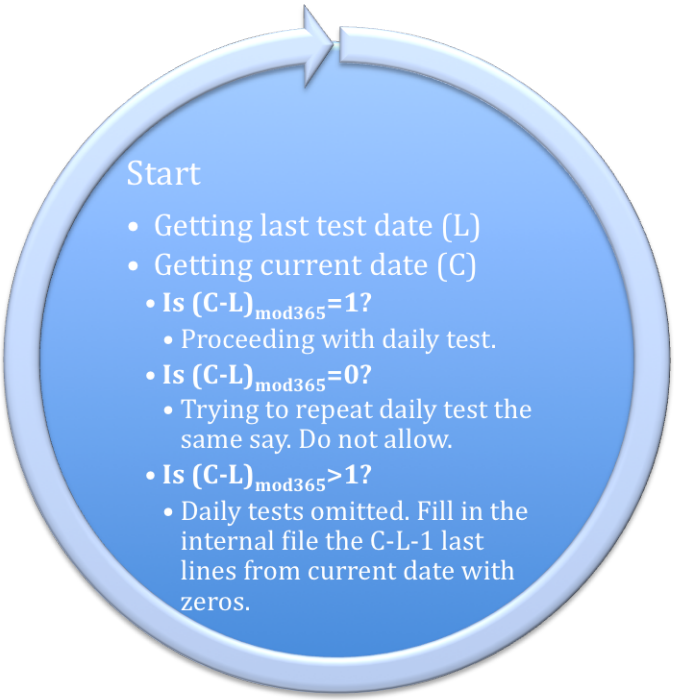

Figure 5.4: Accessing files algorithm

On the other hand, an algorithm is also required to get an access to the correct PEF and FEV1 standard values. As it was mentioned above, standard PEF and FEV1 values are saved into a text file. Starting with the algorithm, first of all AsthmaApp opens the personal patient's data file. Then, age, height and gender information are taken. The files are composed of rows, where each row represents a range of age. Moreover, rows contain a set of values (depending of the height) separate by ";". Finally, the first part of the file corresponds to men values whereas the second to women. Bearing in mind that structure, AsthmaApp takes the patient data in order to get a number, which will be used to seek out the correct value into the file. The method *readLine* from the class *streamReader* is used to read the whole line. Then, the value is read to the next ";", and separate from ";" by using *split* method from *string* class, ensuring a full

reception of the value regardless the number of decimals. At the end, the particular value is obtained and ready to be checked with the results from spirometry.

5.6.2 Taking a Daily Test

Daily test is the AsthmaApp's function which will be probably done more times by the patient. Therefore, it has attempted to design an effective and user-friendly interface, simplifying and avoiding not necessary steps. To perform a successfully daily test, the patient should do the following actions:

- **Answering health questions:** by touching with finger, questions are easily answered.
- **Inputting PEF value:** the application is error-controlled. In the case of introduce letters or values out of range, an error message is shown pleasing to reintroduce the data.
- **Inputting FEV1 value:** as in PEF request, the value is introduced. In this case the range of correct values is significantly different.
- **Wheezes recording:** because of the fact that currently not all the patients will dispose of the recorder, breathing recordings are optional.

After those steps, a final state is displayed to the patient and the information is automatically sent to the server. In order to make the process as easy as possible, if the patient does not have Internet access, the tests results will be saved in a temporary file. Then a button appears in the daily tests menu, allowing the patient to send the last test later, when connection will be available. This feature helps the mobility of the patients and allows them to take tests even without Internet access. It is important to remark that in a case of panic state, the sms alert will be also instantaneously sent. In figure 5.5 is appreciated how the "Sending last test" button appears in the daily test menu after a failed attempt to send the test results to the Hospital Server. Finally, when the information is sent, "Sending last test" button disappears from daily test menu and the file is deleted from the internal phone memory. Algorithm 5.7 shows how the data is sent into a temporary file in a case of Internet connection failure.

Figure 5.5: Pending test results

Algorithm 5.7 Saving pending data

```
private void saveData()
    {
        MessageBox.Show("Error connecting to the server. The results will be saved into a file.", "Error");
        System.IO.FileStream sw = new System.IO.FileStream(Environment.GetFolderPath
        (Environment.SpecialFolder.ApplicationData) + "\\AsthmaAppData\\Test.txt", System.IO.FileMode.
        OpenOrCreate, System.IO.FileAccess.Write, System.IO.FileShare.Read);//Creating temporary file
        System.IO.StreamWriter swrite = new System.IO.StreamWriter(sw);
        swrite.WriteLine("data=1#" + Convert.ToString(sum) + "#" + Convert.ToString(PEF) + "#" +
        Convert.ToString(FEV1) + "####&op=2&user=" + login);//Writing into the file
        swrite.Close();
        sw.Close();
    }
```

5.7 Pollution Alerts Form

Pollution alerts form allows a real-time air quality monitoring, checking the pollution air state in the patient's location. Firstly, the window only displays a button that activates the internal Gps antenna of the smartphone. The coordinates and speed

of the patient are obtained. By means of 3 labels the data are displayed. A timer has been programmed to avoid long searching range waits, and when the time trying to connect is over 40 seconds, a message is displayed saying that the connection has not been possible. Once obtained the data, coordinates are displayed in degrees, minutes and seconds format as well as speed is displayed in km/h. Despite speed is not necessary in order to check the air quality, this functionality is added because it may be useful for the user. The system has been designed as a real-time monitoring, so when the patient changes location, the data is uploaded and displayed automatically. When the data is obtained, another button is available to check the air quality, giving the possibility to send the current coordinates to the Hospital Server, and getting the results of air quality. Finally, a pollution state is displayed on the screen. On the basis that this application feature in implemented to on demand usage, patients are able to check the air quality as many times as they want during the day.

5.8 Data Visualization Forms

Charts are an easy way to visualize the patient's disease evolution. AsthmaApp provides a wide charts section, where the user can check the scores of the questions as well as PEF and FEV1 results. Furthermore, there are 3 options of visualization: last week, last month and last year. By using this powerful tool, the user is able to check the evolution of the disease in a short-period, middle-period or long-period time. Charts are provided by bars, and in each one, morning and evening results from the same day are displayed simultaneously. In questions results, the maximum evening tests score is 12 whereas in morning test is 4 (see also section 5.2.1.4). Morning and evening results in PEF and FEV1 charts are also displayed simultaneously, taking into account that morning levels are around 20% lower.

Windows do not provide any assembly to deal with charts in .Net Compact Framework. Despite there are a lot of material on the web on this issue, the solutions that work for a Compact Framework are commercial with a high price in the market. To solve this problem, a free low-level external library has been used. PocketBarGraph.dll namespace [WZ10] allows the developer to plot a bar chart (bars and axes) with a collection of data introduced. The PockedGraphBar.dll components will be described in the next section.

5.8.1 Using PocketGraphBar.dll

The assembly mainly has the following components:

- GraphMotor: is the controller class that resolves all the logic. It contains the method to input data as well as bars features (color, width, etc.)
- ListGraphs: is a collection of the series of data that it will be plotted.
- ListData: is a collection of the data that it will be plotted.
- GraphPoint: a single key pair value, X and Y where Y depends on the value of X. GraphPoints determine the position in X axis and height of the bars.

In order to ensure a correct plotting of the charts, in the Load event of the form, *GraphMotor* data sources must be filled. Hence, the appearance features are set through the component and the Paint event of the form. At the end, the key for a proper display is to send the System.Windows.Forms.PaintEventArgs of the argument of the form's Paint event to the *GraphMotor*. Another important issue is to use the method *refresh()* in switching different plots to clean the screen and do not superimpose charts.

Algorithm 5.8 Setting chart properties

```
private void Data_Paint(object sender, PaintEventArgs e)
    {
        try
        {//In the load event the the object was filled. Here we only set its properties
           graph1.LeftMargin = 35;
           graph1.LegendFont = new System.Drawing.Font("Arial", 1F, System.Drawing.FontStyle.Regular);
           graph1.AxisColor = Color.White;
           graph1.MaxHeight = 200;
           //The width of each bar
           graph1.Thick = 6;
           //The number of bars displayed for each series of data
           graph1.DisplayTimes = 8;
           graph1.DrawGraphs(e);
        }
        catch (Exception ee)
        {
           MessageBox.Show(ee.ToString());
        }
    }
```

Algorithm 5.9 Filling chart data

```
private void DailyTestCharts_Load(object sender, EventArgs e)
    {//Setting last week graph values
        graph1 = new GraphMotor();
        graph1.Graphs.Add(new ListData());
        graph1.Graphs[0].DisplayColor = Color.Yellow;
        GraphPoint p;
        //Generate data
        for (int i = 6; i >=0; i--)
        {//A new point
            p = new PocketGraphBar.GraphPoint();
            p.X = Convert.ToDecimal(i);
            if ((mresults[((mindex - i) % 365) + (((mindex - i) < 0) ? 365 : 0)] == 0)
            || (mresults[((mindex - i) % 365) + (((mindex - i) < 0) ? 365 : 0)] == -1))
            {
                p.Y = Convert.ToDecimal(0.1);
            }
            else
            {
                p.Y = mresults[((mindex - i) % 365) + (((mindex - i) < 0) ? 365 : 0)];
            }
            graph1.Graphs[0].Add(p);
        }
    }
```

Algorithms 5.9 and 5.8 show how to fill data and set properties to display the last week daily test results. Notice that the vector *mresults* contains all 365 results, corresponding to morning tests (first day of the year in the position 0 and last day of the year in the position 364) and *mindex* is the day of the year of the last test taken. In the *"for"* loop, the last 7 values starting from the last test fill the ListData of the GraphMotor object. Therefore, a mod_{365} is required in each *"if"* condition in order to respect de circular distribution of the results in the vector. As an example, when the last test done is in the second day of the year, the algorithm should take the 2 first values of the vector results as well as the last 5.

5.8.2 Charts Data Processing

As it was introduced in the last section, first of all the application opens morning and evening files and stores the data obtained in 2 vectors: morning and evening results. These vectors contains, depending of the forms: all questions, PEF or FEV1 values stored by AsthmApp. Last week visualization just uses the last 7 test values. Then,

in the cases of last month and last year visualization, 2 other vectors are created. The last month vector uses the last 32 test values and, doing an average of groups of 4 values, obtains 7 values, which will be displayed in the last month chart. On the other hand, the last year vector, by using the last 360 test values, and doing an average of groups of 30 values, obtains 12 final values. Those values corresponds approximately to the last 12 months of the year.

Once obtained the different values, they are inputted to the *GrapthMotor* data sources of the different charts. Finally they are ready to a proper display. The final visualization results are shown in figure 5.6.

Figure 5.6: Last week visualization.

5.9 Edit Patient Information Form

Edit patient information form is displayed in the first run of the program and there is also an access button from the main form to modify the data at any time. In the first run, the welcome form has only a button accessing to edit patient form. When the window is opened, *comboBoxes* and *textBoxes* are blank and all the fields must

be filled in order to access the main form of the application. This way to get data ensures that all the information required before making the first daily test will be previously entered.

On the other hand, in an every day use, there is a button in the main form accessing to edit patient information form. In this case, when the form in opened, AsthmaApp is opening the personal data patient file in order to fill the *textBoxes* with the last information saved into that file. Therefore, the user can see all the fields filled, representing a help to modify some part of the data. Finally, in edit patient form 2 buttons are available: back and apply. Back button closes the form without save the changes whereas apply button takes the data introduced, rewriting the patient personal data file. At last, a message is shown telling that the data have been updated successfully.

Figure 5.7: On the left first run form. On the right edit patient information form.

Figure 5.7 shows the first run of the welcome form as well as a normal edit patient information form. Both forms illustrate the way to introduce and to upload personal data in AsthmaApp.

5.10 Settings Form

Last but not least, the settings form provides 2 important options. The first one allows the user to introduce or modify the login and password of the Hospital Server account. It is interesting to note that each time that the patient starts a daily test, AsthmaApp checks whether the login and password fields are filled. In an affirmative case daily tests starts, otherwise a message is displayed indicating that the patient must fill previously such fields from the settings form. This feature is programmed to avoid problems in sending data to the server, making a more reliable communication.

The second feature consists on chance the application mode. By using a *comboBox*, the patient is able to select between default, children and elderly people modes. Finally, as in the case of edit patient information form, 2 buttons are available (back and apply). Apply button saves the changes introduced by the user rewriting the whole internal data file with the new data introduced. In figure 5.8 a settings form is shown.

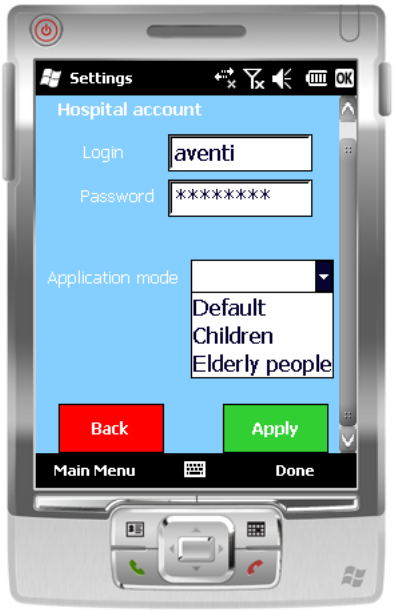

Figure 5.8: Settings form

5.11 Additional AsthmaApp Aspects

AsthmaApp also includes other minor features that make application more manageable and easier to use. Besides, all of them are important to the proper performance of the overall system.

5.11.1 AsthmaApp Icon

In AsthmaApp, an icon has been designed. The icon is a pictorial representation of a medical box, important for the visual identity of the program and it is a shorthand for conveying meaning that the patient perceives almost instantaneously. Despite Visual Studio provides a native icon editor, the icon has been designed externally and then, added to the device project. The following steps show how to add an icon to a Visual C# device project:

- Firstly, the developer has to open AsthmaApp project with Visual Studio.
- In the solution explorer a right-click has to be done and then, select Properties. The project designer appears.
- In the project designer, click the resources tab and then, clicking the Add Resource, drop-down list and select Add New Icon.
- Backing to solution explorer, the icon.ico created previously must be added.
- Finally, in application tab from project designer, the icon added must be selected in the resources application field.

By following those directives, an icon can be added to the application. Furthermore, creating a direct access in the start menu of Smartphone, the application is accessible from the main menu as well as other commercial applications and tools installed previously. Figure 5.9 shows the final result.

Figure 5.9: AsthmaApp icon

5.11.2 Automatic Touchable Keyboard

Another interesting feature introduced by AsthmaApp is the automatically appearance of the touchable keyboard when the *textBoxes* are focused (first touch). By default the user has to touch the *textBox* to get it focused and then, to touch the keyboard button. This improvement allows a more intuitive and fast entering data. To disable the keyboard, it is necessary just to touch the keyboard button or the *"done"* button (also programmed for this functionality) on the right side of the screen.

The software-based input panel (also known as SIP or software keyboard) allows touch-screen enabled devices that do not have a hardware keyboard to simulate keyboard input by using an onscreen touchable keyboard. Within the .NET Compact Framework, the software-based input panel is wrapped up by the *System.Windows.Forms.InputPanel* class, which is present in the *System.Windows.Forms.dll* assembly. In order to interact with the software-based input panel, the developer can drop the InputPanel component from the Visual Studio toolbox onto a form. Then, a reference to the *dll* assembly is added automatically to the code side of the project.

Finally, it is necessary to set the property enabled to true from the inputPanel in all the event-functions gotFocus from each textBox to reach the goal.

5.11.3 Programming Graphics

Despite graphics programming has not been exploited by the application, AsthmaApp deals with graphics when the final health state is displayed. The patient can see on the screen letters with the state written as well as a circle filled with a color corresponding to the state. It is a visual help to make the perception of the state easier.

In order to plot the mentioned circle, the classes *System.Drawing.SolidBrush* and *System.Drawing.Graphics* are used, which are present in the *System.Drawing.dll* assembly. *SolidBrush* class provides a brush that will be used to set the future color of the circle. On the other hand, Graphics class allows to plot several 2-dimension objects such as squares, triangles or ellipses. To plot the health state a particular case of ellipse (circle) is selected. Finally by calling the method dispose, the circle can be displayed onto the screen of the Smartphone. The algorithm 5.10 shows the necessary steps to print a red circle on the center of the screen.

Algorithm 5.10 Plotting a circle

```
using System.Drawing;

private Graphics grap;
private void printCircle(object sender, EventArgs e)
    {
            SolidBrush brush = new SolidBrush(Color.Red);//Creating a red brush
            grap = this.CreateGraphics();
            //Setting position and dimensions
            grap.FillEllipse(brush, new Rectangle(100, 150, 120, 120));
            brush.Dispose();
    }
```

Chapter 6

Conclusions

6.1 Summary and Conclusions

Mobile communications are constantly growing up and asking for a new programming languages and environments. This rapid progress requires the programmer to be aware of new technologies and deal with changing systems and platforms. What is significant is that day by day, new functionalities and improvements are appearing in the mobile communications field, expanding the possibilities of the developer. Meanwhile, already existing applications allow to make the life of the users easier in several aspects, when few years ago it was not possible.

In this study, a Smartphone application have been developed. AsthmaApp is provided with well-known technologies, offering a system able to monitor asthma and COPD patients. Internet connection, bluetooth connection and Gps as well as the use of new medical devices have allowed the patient to treat the disease at home or everywhere. Moreover, a robust programming code has been developed in order to do not lose data, even in the case of non-Internet access. The final purpose of the application is to give a reliable tool, able to replace periodical visits to the doctor as well as to give new facilities, complementing the disease remote monitoring. The experimental programming results demonstrate that the first version of the AsthmaApp is ready and usable. Next months first patients will prove the system and data will start to be collected.

Finally, the study of eHealth solutions in a mobile development could go further and further until find an ideal system to guarantee a full non-risk disease remote monitoring. However, eHealth mobile development is still in an initial phase. In the next years, better and more sophisticate solutions will appear to thread a large group of diseases.

6.2 Future Research

In future work, one of the priorities to improve the application will be performing a better connectivity between medical devices and the smartphone. This fact implies a full bluetooth communication, even to transmit data such as PEF and FEV1 values, which currently are inputted manually. Taking into account that one of the main triggering factors of asthma attack is the increasing stress or physical effort, another future improvement will be to monitor the heart rate of the patient during the daily tests, or even few hours per day, depending on the patient and the degree of the disease.

On the other hand, in order to cover all the market, future work will be also focused on developing the application to other platforms such as Android or iOS (iphone). At the end, one of the limitations of AsthmaApp is that currently is only available in English version. In the next months the application will be translated to polish and then, if it is required, to another languages.

Appendix A. Daily Test Relevant Code

```
namespace AsthmaApp
{

    public partial class DailyTest : Form
    {
        private DateTime dat;
        private int sum = 0;
        private int aux;
        private int row;
        private int column;
        private string sex;
        private decimal PEF;
        private decimal PEFstd;
        private decimal FEV1;
        private decimal FEV1std;
        private int sumtest;
        private decimal result;
        private decimal result2;
        private double final;
        private string path;
        private string path1;
        private string sum2 = "";
        private string login = "";
        private string pass = "";
        private Stream dataStream;
        public bool back;
        private int date;
        private Graphics grap;
        //Bluetooth Passcode: 1234 (IMPORTANT)
        //Listen bluetooth devices availables
        private BluetoothRadio radio;
        //Info about specific device selected
        BluetoothDeviceInfo device;
        //Bluetooth client
        private BluetoothClient cli;
        //Stream with the information from the paired device
        //Way to read data to the device
        private Stream peerStream;
        //Data buffer
        private Stream memstream;
        //Binnary reader
        private BinaryReader reader;
```

```
//Wheezes buffer
private byte[] WBuffer;
bool wheezes = false;
public DailyTest()
{
   InitializeComponent();
   dat = DateTime.Now;
   back = false;
   // GETTING USER NAME AND PASSWORD FROM THE FILE
   System.IO.StreamReader logpass = new System.IO.StreamReader(Environment.GetFolderPath
   (Environment.SpecialFolder.ApplicationData) + "\\AsthmaAppData\\Mode.txt");
   logpass.ReadLine();
   login = logpass.ReadLine();
   pass = logpass.ReadLine();
   logpass.Close();
   if (((login == "")||(login == null)) || ((pass == null)||(pass == "")))
   {
      MessageBox.Show("Please, fill login and password in settings before doing the test", "Error",
      MessageBoxButtons.OK, MessageBoxIcon.Exclamation, MessageBoxDefaultButton.Button2);
      this.Close();
   }

   //Checking last test
   System.IO.FileStream msw = new System.IO.FileStream(Environment.GetFolderPath(Environment.SpecialFolder
   .ApplicationData) + "\\AsthmaAppData\\MTest.txt", System.IO.FileMode.Open, System.IO.FileAccess.Read, System.IO.FileShare.Read);
   System.IO.StreamReader msread = new System.IO.StreamReader(msw);
   date = Convert.ToInt32(msread.ReadLine());
   msread.Close();
   msw.Close();
   if (dat.DayOfYear != 366)//data from day 366 (every 4 years) is not stored, only sent to the server
   {//here we compare the date of last last test done with the curent date
      if (((dat.DayOfYear - date) % 365 + (((dat.DayOfYear - date) < 0) ? 365 : 0)) == 1)
      {
         //Normal situation, last test was 1 day ago
      }
      else if (((dat.DayOfYear - date) % 365 + (((dat.DayOfYear - date) < 0) ? 365 : 0)) == 0)
      {//trying to repeat the test the same day
         MessageBox.Show("Daily test done, do the next one tomorrow", "Error", MessageBoxButtons.OK,
         MessageBoxIcon.Exclamation, MessageBoxDefaultButton.Button2);
         this.Close();
      }
      else
      {//tests missed, now we have to fill with 0's as days as the patient missed
         System.IO.FileStream msw2 = new System.IO.FileStream(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
         + "\\AsthmaAppData\\MTest.txt", System.IO.FileMode.Open, System.IO.FileAccess.ReadWrite, System.IO.FileShare.Read);
         System.IO.StreamReader read = new System.IO.StreamReader(msw2);
         int cont = 0;
         int linemove = 0;

         bool start = false;
         for (int i = 0; i <=date; i++)//Getting the position for start writing
         {
            cont = cont + read.ReadLine().Length + 2;
            linemove++;
         }
         msw2.Position = cont;
         System.IO.StreamWriter write = new System.IO.StreamWriter(msw2);
         for (int i = 0; i < (((dat.DayOfYear - date) % 365 + (((dat.DayOfYear - date) < 0) ? 365 : 0)) - 1); i++)//Filling the string for fix the line length
         {
            if (linemove == 366)
```

```
            {
                start = true;
                cont = i;
                break;
            }
            write.WriteLine("-1;-1;-1;0000000000;");
            linemove++;
        }
        write.Close();
        read.Close();
        msw2.Close();

        if (start == true)//here we start from the begging filling if it's neccessary
        {//cyclic behaviour
            System.IO.FileStream msw22 = new System.IO.FileStream(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
            + "\\AsthmaAppData\\MTest.txt", System.IO.FileMode.Open, System.IO.FileAccess.Write, System.IO.FileShare.Read);
            msw22.Position = 5;
            System.IO.StreamWriter write22 = new System.IO.StreamWriter(msw22);
            for (int i = cont; i < (((dat.DayOfYear - date) % 365 + (((dat.DayOfYear - date) < 0) ? 365 : 0)) - 1); i++)
            {
                write22.WriteLine("-1;-1;-1;0000000000;");
            }
            write22.Close();
            msw22.Close();
        }

        System.IO.FileStream msw3 = new System.IO.FileStream(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
        + "\\AsthmaAppData\\MTest.txt", System.IO.FileMode.Open, System.IO.FileAccess.Write, System.IO.FileShare.Read);
        System.IO.StreamWriter write2 = new System.IO.StreamWriter(msw3);
        //Here we write the actual date -1 (before test) mod 365
        if ((dat.DayOfYear - 1) == 0)
        {
            write2.WriteLine("365");
        }
        else
        {
            if ((dat.DayOfYear - 1) < 10)
            {
                write2.WriteLine("00" + Convert.ToString(dat.DayOfYear - 1));//fix longitudes
            }
            else if (((dat.DayOfYear - 1) >= 10) && ((dat.DayOfYear - 1) < 100))
            {
                write2.WriteLine("0" + Convert.ToString(dat.DayOfYear - 1));
            }
            else
            {
                write2.WriteLine(Convert.ToString(dat.DayOfYear - 1));
            }

        }
        write2.Close();
        msw3.Close();
    }
  }
}
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    sum2 = comboBox1.Text;
}
```

```
private void button2_Click(object sender, EventArgs e)
{
    if (sum2 == "")
    {
        MessageBox.Show("Please, answer the question", "Error");
    }
    else//Let's correlate with times that the patient took medicine
    {
        aux = Convert.ToInt32(sum2);
        if ((sum - aux) >= 2)
        {
            aux = System.Math.Abs(sum - aux);
            aux--;
            if ((sum - aux) <= 0)
            {
                sum = 0;
            }
            else
            {
                sum = sum - aux;
            }
        }
        if ((sum - aux) <= -2)
        {
            aux = System.Math.Abs(sum - aux);
            aux--;
            if ((sum + aux) >= 4)
            {
                sum = 4;
            }
            else
            {
                sum = sum + aux;
            }
        }
        //Here we change values for better visualization:
        //4 points=green, 0 points=panic
        if (sum == 4)
        {
            sum = 0;
        }
        else if (sum == 3)
        {
            sum = 1;
        }
        else if (sum == 2)
        {
            sum = 2;
        }
        else if (sum == 1)
        {

            sum = 3;
        }
        else
        {
            sum = 4;
        }
        label1.Visible = false;
        label3.Visible = false;
        label4.Visible = false;
        comboBox1.Visible = false;
        button2.Visible = false;
        label5.Visible = true;
```

```
            textBox1.Visible = true;
            button1.Visible = true;
        }
    }

    private void menuItem1_Click(object sender, EventArgs e)
    {
        if (inputPanel1.Enabled == true)
        {
            inputPanel1.Enabled = false;
        }

        this.Close();
    }

    private void textBox1_TextChanged(object sender, EventArgs e)
    {
        try
        {
            PEF = Convert.ToDecimal(textBox1.Text);
        }
        catch (Exception) { MessageBox.Show("Please, enter a correct value", "Error"); }
    }

    private void textBox1_GotFocus(object sender, EventArgs e)
    {
        if (textBox1.Visible == true)
        {
            inputPanel1.Enabled = true;
        }
    }

    private void textBox2_TextChanged(object sender, EventArgs e)
    {
        try
        {
            FEV1 = Convert.ToDecimal(textBox2.Text);
        }
        catch (Exception) {MessageBox.Show("Please, enter a correct value", "Error"); }
    }

    private void textBox2_GotFocus(object sender, EventArgs e)
    {
        if (textBox2.Visible == true)
        {
            inputPanel1.Enabled = true;
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {


        if ((PEF<1)||(PEF>2000))
        {
            MessageBox.Show("Please, enter a correct value", "Error");
        }
        else
        {
            label5.Visible = false;
            textBox1.Visible = false;
            button1.Visible = false;
            label6.Visible = true;
            textBox2.Visible = true;
            button3.Visible = true;
```

```
    }
}

private void button3_Click(object sender, EventArgs e)
{
    if ((FEV1 < -5) || (FEV1 >= 10))
    {
        MessageBox.Show("Please, enter a correct value", "Error");
    }
    else
    {//Openning PEF_data file
        path = System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().GetName().CodeBase);
        path1 = System.IO.Path.Combine(path, "PEF_data.txt");
        //path = path + "\\PEF_data.csv";
        System.IO.File.OpenRead(path1);
        System.IO.StreamReader sread = new System.IO.StreamReader(path1);
        for (int i = 0; i < (row - 1); i++)
        {
            sread.ReadLine();
        }
        // reusing string sex
        sex = sread.ReadLine();
        string[] b = sex.Split(new char[1] { ',' });
        PEFstd = Convert.ToDecimal(b[column - 1]);
        sread.Close();
        //this.Text = Convert.ToString(PEFstd);

        //Openning FEV1_data file
        path1 = System.IO.Path.Combine(path, "FEV1_data.txt");
        System.IO.File.OpenRead(path1);
        System.IO.StreamReader sread1 = new System.IO.StreamReader(path1);
        for (int i = 0; i < (row - 1); i++)
        {
            sread1.ReadLine();
        }
        // reusing string sex
        sex = sread1.ReadLine();
        string[] c = sex.Split(new char[1] { ',' });
        FEV1std = Convert.ToDecimal(c[column - 1]);
        sread1.Close();
        //Setting final result
        PEFstd = System.Decimal.Multiply(PEFstd, 0.8m);
        result = System.Math.Abs(PEF - PEFstd) / PEFstd;
        result = result * 100;
        FEV1std = System.Decimal.Multiply(FEV1std, 0.8m);
        result2 = System.Math.Abs(FEV1 - FEV1std) / FEV1std;
        result2 = result2 * 100;
        result = System.Decimal.Multiply(result, 0.5m) + System.Decimal.Multiply(result2, 0.5m);
        //this.Text = Convert.ToString(result);
        if (result <= 15)
        {

            sumtest = 4;
        }
        else if ((result > 15) && (result <= 30)) //green state test
        {
            sumtest = 3;
        }
        else if ((result > 30) && (result <= 40)) //yellow state test
        {
            sumtest = 2;
        }
        else if ((result > 40) && (result <= 60)) //red state test
        {
            sumtest = 1;
```

```
        }
        else //panic state test
            sumtest = 0;
        final = sum * 0.4 + sumtest * 0.6;
        Refresh();
        //Displaying the final state
        if (final > 3)//Green
        {
            label9.Visible = true;
            SolidBrush brush = new SolidBrush(Color.LawnGreen);
            grap = this.CreateGraphics();
            grap.FillEllipse(brush, new Rectangle(100, 150, 120, 120));
            brush.Dispose();
        }
        else if ((final > 2)&&(final <= 3))//Yellow
        {
            label9.Text = "Daily test state: yellow";
            label9.ForeColor = Color.FromArgb(255, 255, 128);
            label9.Visible = true;
            SolidBrush brush = new SolidBrush(Color.FromArgb(255, 255, 128));
            grap = this.CreateGraphics();
            grap.FillEllipse(brush, new Rectangle(100, 150, 120, 120));
            brush.Dispose();
        }
        else if ((final > 1) && (final <= 2))//Red
        {
            label9.Text = "Daily test state: red";
            label9.ForeColor = Color.Red;
            label9.Visible = true;
            SolidBrush brush = new SolidBrush(Color.Red);
            grap = this.CreateGraphics();
            grap.FillEllipse(brush, new Rectangle(100, 150, 120, 120));
            brush.Dispose();
        }
        else//Panic
        {
            label9.Text = "Daily test state: panic";
            label9.ForeColor = Color.Black;
            label9.Visible = true;
            SolidBrush brush = new SolidBrush(Color.Black);
            grap = this.CreateGraphics();
            grap.FillEllipse(brush, new Rectangle(100, 150, 120, 120));
            brush.Dispose();
            try//Sending sms alert
            {
                SmsMessage sms = new SmsMessage();//Creating a new smsMessage object
                sms.Body = "The patient"+login+"has an asthma attack risk!!";//Message text
                sms.To.Add(new Recipient("783614210"));//Here it should be the doctor's mobile
                sms.Send();
                MessageBox.Show("Alarm sent to the doctor");
            }
            catch (Exception)//Error control


            {
                MessageBox.Show("Error sending sms alert");
            }
        }
        button4.Visible = true;
    }
}

private void menuItem2_Click(object sender, EventArgs e)
{
    inputPanel1.Enabled = false;
}
```

```
private void button4_Click(object sender, EventArgs e)
{
    button4.Visible = false;
    label9.Visible = false;
    grap.Clear(Color.LightSkyBlue);
    grap.Dispose();
    Refresh();
    //Wheezes detection question
    label12.Visible = true;
    button7.Visible = true;
    button8.Visible = true;
}
private void button7_Click(object sender, EventArgs e)
{
    label12.Visible = false;
    button7.Visible = false;
    button8.Visible = false;
    //Wheezes detection
    label8.Visible = true;
    label10.Visible = true;
    button5.Visible = true;
    wheezes = true;
}
private void button8_Click(object sender, EventArgs e)
{//Test without wheezes detection
    label12.Visible = false;
    button7.Visible = false;
    button8.Visible = false;
    sendData();
}

private void button5_Click(object sender, EventArgs e)
{
    label10.Visible = false;
    button5.Visible = false;
    label11.Visible = true;
    comboBox2.Visible = true;
    cli = new BluetoothClient();
    memstream = new MemoryStream();
    radio = BluetoothRadio.PrimaryRadio;//Selecting bluetooth radio
    if (radio == null)
    {//In case of incompatibility
        MessageBox.Show("No support Bluetooth radio/stack found.");
    }
    else if (radio.Mode != InTheHand.Net.Bluetooth.RadioMode.Discoverable)
    {
        radio.Mode = RadioMode.Discoverable;//Ensuring discoverable mode
    }

    BluetoothDeviceInfo[] info = cli.DiscoverDevices(255, true, false, true);//In the list appear paired and discoverable devices
    comboBox2.DataSource = info;//Assigning the data to combobox
    comboBox2.DisplayMember = "DeviceName";


}
private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
{//Selecting a device in the list
    device = comboBox2.SelectedItem as BluetoothDeviceInfo;
    button6.Visible = true;
}
private void button6_Click(object sender, EventArgs e)
{
    button6.Visible = false;
    comboBox2.Visible = false;
    label11.Visible = false;
    label8.Visible = false;
```

```
    //Establishing communication
    BluetoothAddress addr = device.DeviceAddress;
    BluetoothEndPoint ep = new BluetoothEndPoint(addr, BluetoothService.SerialPort);//Setting bluetooth protocol
    try
    {
        cli.Connect(ep);
        peerStream = cli.GetStream();//Getting the string, a bidirectional communication is allowed
    }
    catch (Exception)
    {
        // Couldn't connect.
        MessageBox.Show("Could not connect to the device " + device.DeviceName + ". Please retry the test later");
        Dispose(false);
        this.Close();
    }
    readData();//Then, the part of code for read the data in executed
}
 private void readData()
{
    //First of all, we will read the stream in block of 1024 bytes
    reader = new BinaryReader(peerStream);
    byte[] buffer = new byte[1024];
    int bytesRead = 0;
    try
    {
        while ((bytesRead = reader.Read(buffer, 0, buffer.Length)) != 0)
        {
            memstream.Write(buffer, 0, bytesRead);
        }
    }
    catch (System.IO.EndOfStreamException)
    {
        // Lost the connection.  So, we're disconnected now.
        MessageBox.Show("Connection lost during transaction. Please, retry the test later");
        Dispose(false);
        this.Close();
    }
    WBuffer = new byte[memstream.Length];
    memstream.Write(WBuffer, 0, WBuffer.Length);
    memstream.Close();
    peerStream.Close();
    cli.Close();
    MessageBox.Show("File transmited successfully. Start sending results to the server");
    sendData();
}
private void sendData()
{
    // CONNECTION TO THE SERVER (daily test result)
    // Create a request using a URL that can receive a post.
    WebRequest request = WebRequest.Create("http://amonit.kt.agh.edu.pl/astma/index.php");
    // Set the Method property of the request to POST.

    request.Method = "POST";
    // Create POST data and convert it to a byte array.
    //Set the string which will be sent to the server as a identifier
    string postData = "log=" + login + "&pas=" + pass;
    byte[] byteArray = Encoding.UTF8.GetBytes(postData);
    // Set the ContentType property of the WebRequest.
    request.ContentType = "application/x-www-form-urlencoded";
    // Set the ContentLength property of the WebRequest.
    request.ContentLength = byteArray.Length;
    // Get the request stream.
    try
    {
        dataStream = request.GetRequestStream();
```

```
}
catch (WebException)
{
    saveData();
}
// Write the data to the request stream.
dataStream.Write(byteArray, 0, byteArray.Length);
// Close the Stream object.
dataStream.Close();
// Get the response.
WebResponse response = request.GetResponse();
// Open the stream using a StreamReader for easy access.
StreamReader reader = new StreamReader(response.GetResponseStream(), Encoding.UTF8);
// Read the content.
string responseFromServer = reader.ReadToEnd();
reader.Close();
if (responseFromServer == "<html><body>Auth err</body></html>")
{
    MessageBox.Show("Authentication error, please verify your login and password in settings section. Then retry the test.", "Connection error");
    reader.Close();
    dataStream.Close();
    response.Close();
}
else
{
    reader.Close();
    response.Close();
    WebRequest request2 = WebRequest.Create("http://amonit.kt.agh.edu.pl/astma/data.php");
    // Set the Method property of the request to POST.
    request2.Method = "POST";
    //1 for daily test morning, 2 for evening, 3 on demand
    postData = "data=1#" + Convert.ToString(sum) + "#" + Convert.ToString(PEF) + "#" + Convert.ToString(FEV1) + "####&op=2&user=" + login;
    byte[] byteArray2 = Encoding.UTF8.GetBytes(postData);
    // Set the ContentType property of the WebRequest.
    request2.ContentType = "application/x-www-form-urlencoded";
    // Set the ContentLength property of the WebRequest.
    request2.ContentLength = byteArray2.Length;
    try
    {
        dataStream = request2.GetRequestStream();
    }
    catch (WebException)
    {
        saveData();
    }
    dataStream.Write(byteArray2, 0, byteArray2.Length);
    dataStream.Close();
    WebResponse response2 = request2.GetResponse();
    response2.Close();
    if (wheezes == true)
    {

        //SEND DATA TO THE SERVER (Wheezes recording)
        WebRequest request3 = WebRequest.Create("http://amonit.kt.agh.edu.pl/astma/index.php");
        // Set the Method property of the request to POST.
        request3.Method = "POST";
        // Create POST data and convert it to a byte array.
        //Set the string which will be sent to the server as a identifier
        string postData3 = "log=" + login + "&pas=" + pass;
        byte[] byteArray3 = Encoding.UTF8.GetBytes(postData3);
        // Set the ContentType property of the WebRequest.
        request3.ContentType = "application/x-www-form-urlencoded";
        // Set the ContentLength property of the WebRequest.
        request3.ContentLength = byteArray3.Length;
```

```
// Get the request stream.
try
{
   dataStream = request3.GetRequestStream();
}
catch (WebException)
{
   MessageBox.Show("Error connecting to the server. Please be sure that internet access is available and retry later", "Error");
   Dispose(false);
   this.Close();
}
// Write the data to the request stream.
dataStream.Write(byteArray3, 0, byteArray3.Length);
// Close the Stream object.
dataStream.Close();
// Get the response.
WebResponse response3 = request3.GetResponse();
response3.Close();
WebRequest request4 = WebRequest.Create("http://amonit.kt.agh.edu.pl/astma/data.php");
// Set the Method property of the request to POST.
request4.Method = "POST";
string postData4 = "data=5####&op=2&user=" + login;
byte[] byteArray4 = Encoding.UTF8.GetBytes(postData4);
// Set the ContentType property of the WebRequest.
request4.ContentType = "application/x-www-form-urlencoded";
// Set the ContentLength property of the WebRequest.
request4.ContentLength = byteArray4.Length;
try
{
   dataStream = request4.GetRequestStream();
}
catch (WebException)
{
   MessageBox.Show("Error connecting to the server. Please be sure that internet access is available and retry later", "Error");
   Dispose(false);
   this.Close();
}
dataStream.Write(byteArray4, 0, byteArray4.Length);
dataStream.Close();
WebResponse response4 = request4.GetResponse();
response4.Close();
WebRequest request5 = WebRequest.Create("http://amonit.kt.agh.edu.pl/astma/upload.php");
string boundary = "----------------------------" + DateTime.Now.Ticks.ToString("x");
// Set the Method property of the request to POST.
request5.Method = "POST";
// Set the ContentType property of the WebRequest.
request5.ContentType = "multipart/form-data; boundary=" + boundary;
string postData5 = "\r\n--" + boundary + "\r\n Content-Disposition: form-data; name=MAX_FILE_SIZE\r\n\r\n50000000\r\n--" +
boundary + "\r\nContent-Disposition: form-data; name=oopp\r\n\r\n251\r\n--" + boundary + "\r\nContent-Disposition: form-data;
name=userfile; filename=recording.wav\r\nContent-Type: audio/wav\r\n\r\n";
byte[] byteArray5 = Encoding.UTF8.GetBytes(postData5);


byte[] endBoundaryBytes = System.Text.Encoding.UTF8.GetBytes("\r\n--" + boundary + "--\r\n");
// Set the ContentLength property of the WebRequest.
request5.ContentLength = byteArray5.Length + WBuffer.Length + endBoundaryBytes.Length;
// Get the request stream.
try
{
   dataStream = request5.GetRequestStream();
}
catch (WebException)
{
   MessageBox.Show("Error connecting to the server. Please be sure that internet access is available and retry later", "Error");
   Dispose(false);
   this.Close();
```

```
            }
            dataStream.Write(byteArray5, 0, byteArray5.Length);
            // Write the data to the request stream.
            dataStream.Write(WBuffer, 0, WBuffer.Length);
            //Write the trailing boundary
            dataStream.Write(endBoundaryBytes, 0, endBoundaryBytes.Length);
            // Close the Stream object.
            dataStream.Close();
            // Get the response.
            WebResponse response5 = request5.GetResponse();
            response5.Close();
        }
        MessageBox.Show("Results sent successfully");
        //END Connection
        //Saving the info into a file
        if (dat.DayOfYear != 366)
        {
            System.IO.FileStream msw = new System.IO.FileStream(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
            + "\\AsthmaAppData\\MTest.txt", System.IO.FileMode.Open, System.IO.FileAccess.ReadWrite, System.IO.FileShare.Read);
            System.IO.StreamReader msread = new System.IO.StreamReader(msw);
            int cont = 0;
            for (int i = 0; i < dat.DayOfYear; i++)//Getting the position for start writing
            {
                cont = cont + msread.ReadLine().Length + 2;
            }
            msw.Position = cont;//Here we move the string pointer to the correct position for write
            //Setting the string with useful data
            string text = Convert.ToString(sum) + ";" + Convert.ToString(PEF) + ";" + Convert.ToString(FEV1) + ";";
            string zeros = "";
            for (int i = text.Length; i < 19; i++)//Filling the string for fix the line length
            {
                zeros = zeros + "0";
            }
            System.IO.StreamWriter mswrite = new System.IO.StreamWriter(msw);
            mswrite.WriteLine(text + zeros + ";");//Writting line
            mswrite.Close();
            msread.Close();
            msw.Close();
            System.IO.FileStream msw2 = new System.IO.FileStream(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
            + "\\AsthmaAppData\\MTest.txt", System.IO.FileMode.Open, System.IO.FileAccess.Write, System.IO.FileShare.Read);
            System.IO.StreamWriter write = new System.IO.StreamWriter(msw2);
            if (dat.DayOfYear < 10)
            {
                write.WriteLine("00" + Convert.ToString(dat.DayOfYear));//fix longitudes, always 3 numbers
            }
            else if ((dat.DayOfYear >= 10) && (dat.DayOfYear < 100))
            {
                write.WriteLine("0" + Convert.ToString(dat.DayOfYear));
            }
            else
            {

                write.WriteLine(Convert.ToString(dat.DayOfYear));
            }
            write.Close();
            msw2.Close();
        }
    }
    //Closing form
    back = true;//this variable is for back to the main form, not to daily test menu
    this.Close();
}
private void saveData()
{
```

```
            MessageBox.Show("Error connecting to the server. The results will be saved into a file.", "Error";
            System.IO.FileStream sw = new System.IO.FileStream(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
            + "\\AsthmaAppData\\Test.txt", System.IO.FileMode.OpenOrCreate, System.IO.FileAccess.Write, System.IO.FileShare.Read);
            System.IO.StreamWriter swrite = new System.IO.StreamWriter(sw);
            swrite.WriteLine("data=1#" + Convert.ToString(sum) + "#" + Convert.ToString(PEF) + "#" + Convert.ToString(FEV1) + "####&op=2&user=" + login);
            swrite.Close();
            sw.Close();
            //Saving the info into a file
            if (dat.DayOfYear != 366)
            {
                System.IO.FileStream msw = new System.IO.FileStream(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
                + "\\AsthmaAppData\\MTest.txt", System.IO.FileMode.Open, System.IO.FileAccess.ReadWrite, System.IO.FileShare.Read);
                System.IO.StreamReader msread = new System.IO.StreamReader(msw);
                int cont = 0;
                for (int i = 0; i < dat.DayOfYear; i++)//Getting the position for start writing
                {
                    cont = cont + msread.ReadLine().Length + 2;
                }
                msw.Position = cont;//Here we move the string pointer to the correct position for write
                //Setting the string with useful data
                string text = Convert.ToString(sum) + ";" + Convert.ToString(PEF) + ";" + Convert.ToString(FEV1) + ";";
                string zeros = "";
                for (int i = text.Length; i < 19; i++)//Filling the string for fix the line length
                {
                    zeros = zeros + "0";
                }
                System.IO.StreamWriter mswrite = new System.IO.StreamWriter(msw);
                mswrite.WriteLine(text + zeros + ";");//Writting line
                mswrite.Close();
                msread.Close();
                msw.Close();
                System.IO.FileStream msw2 = new System.IO.FileStream(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
                + "\\AsthmaAppData\\MTest.txt", System.IO.FileMode.Open, System.IO.FileAccess.Write, System.IO.FileShare.Read);
                System.IO.StreamWriter write = new System.IO.StreamWriter(msw2);
                if (dat.DayOfYear < 10)
                {
                    write.WriteLine("00" + Convert.ToString(dat.DayOfYear));//fix longitudes, always 3 numbers
                }
                else if ((dat.DayOfYear >= 10) && (dat.DayOfYear < 100))
                {
                    write.WriteLine("0" + Convert.ToString(dat.DayOfYear));
                }
                else
                {
                    write.WriteLine(Convert.ToString(dat.DayOfYear));
                }
                write.Close();
                msw2.Close();
            }
            Dispose(false);
            this.Close();
        }

    }
}
```

Appendix B. Main Form Relevant Code

```
namespace AsthmaApp
{
   public partial class MainForm : Form
   {
      private bool cond = true;
      public int mode=0;
      private DateTime dat;

      public MainForm()
      {
         InitializeComponent();
      }

      private void menuItem1_Click(object sender, EventArgs e)
      {
         this.Close();
      }

      private void button4_Click(object sender, EventArgs e)
      {
         PatientInfo info;
         info = new PatientInfo();
         info.ShowDialog();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         DailyTestMenu men;
         men = new DailyTestMenu(mode);
         men.ShowDialog();
      }

      private void button3_Click(object sender, EventArgs e)
      {
         Settings set;
         set = new Settings(mode);
         set.ShowDialog();
      }

      private void MainForm_Activated(object sender, EventArgs e)
      {
         if (cond == true)
         {
            dat = DateTime.Now;
            cond = false;
            Welcome first;
            first = new Welcome();
            if (System.IO.Directory.Exists(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\AsthmaAppData") == false)
            {
               System.IO.Directory.CreateDirectory(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\AsthmaAppData");
```

```
first.label2.Visible = true;
first.label4.Visible = false;
first.button1.Visible = true;
System.IO.FileStream sw = new System.IO.FileStream(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
+ "\\AsthmaAppData\\Mode.txt", System.IO.FileMode.OpenOrCreate, System.IO.FileAccess.Write, System.IO.FileShare.Read);
System.IO.StreamWriter swrite = new System.IO.StreamWriter(sw);
swrite.WriteLine("0");
swrite.Close();
sw.Close();
//Inicialize morning test file
System.IO.FileStream msw = new System.IO.FileStream(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
+ "\\AsthmaAppData\\MTest.txt", System.IO.FileMode.OpenOrCreate, System.IO.FileAccess.Write, System.IO.FileShare.Read);
System.IO.StreamWriter mswrite = new System.IO.StreamWriter(msw);
if ((dat.DayOfYear - 1) == 0)
{
    mswrite.WriteLine("365");
}
else
{
    if ((dat.DayOfYear-1) < 10)
    {
        mswrite.WriteLine("00" + Convert.ToString(dat.DayOfYear-1));//fix longitudes
    }
    else if (((dat.DayOfYear-1) >= 10) && ((dat.DayOfYear-1) < 100))
    {
        mswrite.WriteLine("0" + Convert.ToString(dat.DayOfYear-1));
    }
    else
    {
        mswrite.WriteLine(Convert.ToString(dat.DayOfYear-1));
    }
}
for (int i = 0; i < 365; i++)
{
    mswrite.WriteLine("-1;-1;-1;0000000000;");
}
mswrite.Close();
msw.Close();
//Inicialize evening test file
System.IO.FileStream esw = new System.IO.FileStream(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
+ "\\AsthmaAppData\\ETest.txt", System.IO.FileMode.OpenOrCreate, System.IO.FileAccess.Write, System.IO.FileShare.Read);
System.IO.StreamWriter eswrite = new System.IO.StreamWriter(esw);
if ((dat.DayOfYear - 1) == 0)
{
    eswrite.WriteLine("365");
}
else
{
    if ((dat.DayOfYear-1) < 10)
    {
        eswrite.WriteLine("00" + Convert.ToString(dat.DayOfYear-1));//fix longitudes

    }
    else if (((dat.DayOfYear-1) >= 10) && ((dat.DayOfYear-1) < 100))
    {
        eswrite.WriteLine("0" + Convert.ToString(dat.DayOfYear-1));
    }
    else
    {
        eswrite.WriteLine(Convert.ToString(dat.DayOfYear-1));
    }
}
for (int j = 0; j < 365; j++)
{
```

```
                eswrite.WriteLine("-1;-1;-1;0000000000;");
            }
            eswrite.Close();
            esw.Close();
        }
        first.ShowDialog();
    }
    System.IO.StreamReader sread = new System.IO.StreamReader(Environment.GetFolderPath
    (Environment.SpecialFolder.ApplicationData) + "\\AsthmaAppData\\Datapatient.txt");
    label1.Text = sread.ReadLine();
    sread.Close();
    System.IO.StreamReader sr = new System.IO.StreamReader(Environment.GetFolderPath
    (Environment.SpecialFolder.ApplicationData) + "\\AsthmaAppData\\Mode.txt");
    mode = Convert.ToInt32(sr.ReadLine());
    sr.Close();
}
private void button5_Click(object sender, EventArgs e)
{
    Pollution_Alert poll;
    poll = new Pollution_Alert();
    poll.ShowDialog();
}
private void button2_Click(object sender, EventArgs e)
{
    Graph charts;
    charts = new Graph();
    charts.ShowDialog();
}
}
}
```

Bibliography

[BPT06] K. A. Banitsas, K. Perakis, S. Tachakra, D. Koutsouris, "Using 3G links to develop a teleconsultation system between a moving ambulance and an A & E basestation", International Journal of Telemedicine and Telecare, Vol. 12, pp 23-25, Royal society of Medicine press, 2006.

[BTM06] Braunstein B, Trimble T, Mishra R, Manoj BS, Rao R, Lenert L, "Feasibility of using distributed Wireless Mesh Networks for medical emergency response", Proc. Annual Symposium American Med. Informatics Assoc. 2006

[CHL06] Hsueh-Ting Chu; Chir-Chang Huang; Zhi-Hui Lian; Tsai, J.J.P.; "A ubiquitous warning system for asthma-inducement" Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on Volume: 2 Publication Year: 2006 , Page(s): 186 - 191

[CIM07] R. A Clark, S. C Inglis, F. A McAlister, J. G F Cleland, S. Stewart, "Telemonitoring or struc-tured telephone suport programmes for patients with chronic heart failure: systematic review and meta-analysis" BMJ, (2007)

[D07] S. Drude, "Requirements and Application Scenarios for Body Area Networks," Mobile and Wireless Communications Summit, 2007. 16th IST , vol., no., pp.1-5, 1-5 July 2007

[DDS09] T. O'Donovan, J. O'Donoghue, C. Sreenan, P. O'Reilly, D. Sammon, K. O'Connor, "A Context Aware Wireless Body Area Network (BAN)", In proceedings of the Pervasive Health Conference 2009. (2009)

[GMG09] A. Gobbi, I. Milesi, L. Govoni, A. Pedotti, R. L. Dellaca', "A new telemedicine system for the home monitoring of lung function in patients with obstructive respiratory diseases", International Conference on eHealth, Telemedicine, and Social Medicine (2009) 119-122

[JJP09] K. Jeong, E. Jung, D. K. Park „Trends of wireless u-Health”, The International Symposium on Communication and Information Technologies (2009) 829-833

[KCT09] Zeashan H. Khan, Denis G. Catalot and J.M. Thiriet, "Wireless Network Architecture for Diagnosis and Monitoring Applications", MASAUM Journal of Computing, Volume 1 Issue 2, September 2009

[MT10] Jacek Matulewski, Bartosz Turowski, “Programowanie aplikacji dla urządzeń mobilnych z systemem Windows Mobile”, Helion 2010

[P09] O. Postolache at al., “Indoor Monitoring of Respiratory Distress Triggering Factor Using a Wireless Sensing Network and a Smart Phone”, Int. Instrumentation and Measurement Technology Conf., Sin-gapore (2009)

[PBK05] K. Perakis, K. Banitsas, G. Konnis, D. Koustsouris, “3G networks in emergency telemedicine – an in-depth evaluation & analysis”, 27th Annual International Conference of the Engineering in Medicine and Biology Society, 2005

[PGX2009] Hung Keng Pung, Tao Gu, Wenwei Xue, Paulito P. Palmes, Jian Zhu, Wen Long Ng, Chee Weng Tang, Nguyen Hoang Chung, “Context-Aware Middleware for Pervasive Elderly Homecare”, 2009.

[PSD09] Putra E.H., Supriyanto E., Din J., Satria H., "Cross Layer Design of Wireless LAN for Telemedicine Application", Third Asia International Conference on Modelling & Simulation, 2009

[QBB09] Bao Quach, Manikanden Balakrishnan, Driss Benhaddou, Xiaojing Yuan, "Implementation of integrated wireless health monitoring network", International Symposium on Mobile Ad Hoc Networking & Computing, New Orleans, Louisiana, USA, May 18, 2009

[SN05] N. Shenoy, H. Nazeran, „A PDA-based Network for Telemonitoring Asthma Triggering Gases In the El Paso School Districts of the US – Mexico Border region.” IEEE Eng. in Medicine and Biology Conf., Shanghai, China, September 1-4,(2005) 5186-5189

[T08] Sing-Hui Toh at al., “WSN Based Personal Mobile Physiological Monitoring and Management System for Chronic Disease”, Int. Conf. on Convergence and Hybrid Information Technology (2008) 467-472

[TMS] A. Talukder, S. Monacos, Tanwir Sheikh, "Distributed MuItisensor Processing and CIassification under Constrained Resourcesfor Mobile Health Monitoring and Remote Environmental Monitoring".

[TWA09] Andrea Taylor, Richard Wilson, Stefan Agamanolis, "A Home Health Monitoring System Designed to Support Carers in Their Caring Role", 2009.

[Web0] http://www.who.int/topics/asthma/en/

[Web1] www.spirometry.com/ENG/Products/spirobank2.asp

[Web2] http://msdn.microsoft.com

[Web3] Bluetooth library, http://inthehand.com/content/32feet.aspx

[Web4] Charts assembly, http://www.codeproject.com/KB/windows/pocketbargraph.aspx

[Web5] IEEE P802.15 Wireless Next Generation Standing Committee (SCwng) for Wireless Personal Area Networks – http://www.ieee802.org/15/pub/SCwng.html

[Web6] Http POST, http://technet.rapaport.com/Info/LotUpload/SampleCode/Full_Example.aspx

[WebPjct] https://www.iip.net.pl/en/project

[WZ10] Wiśniewski, M. Zieliński, T. "Digital analysis methods of wheezes in asthma" ICSES'10, s.69–72 2010